WebSphere® Application Server—Express for Distributed Platforms, Version 6.1

IBM

Administering applications and their environment

Note

Before using this information, be sure to read the general information under “Notices” on page 2339.

Compilation date: May 18, 2006

Contents

How to send your comments

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

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-0206.

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

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

Chapter 1. Overview and new features for administering applications and their environments

Use the links provided in this topic to learn about the administrative features.

What is new for administrators

This topic provides an overview of new and changed features of system administration.

"Introduction: System administration"

This topic describes the administration of the product and the applications that runs on it.

See also "Introduction: Environment" on page 5 and "Variables" on page 6.

Introduction: System administration

Note:  If you would prefer to browse PDF versions of this documentation using your Adobe Reader, see the **System Administration** PDF files available from www.ibm.com/software/webservers/appserv/infocenter.html.

A variety of tools are provided for administering the WebSphere Application Server product:

- **Console**

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

  For more information, refer to "Introduction: Administrative console."
- **Scripting**

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

  For more information, refer to "Introduction: Administrative scripting (wsadmin)" on page 2.
- **Commands**

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

  For more information, refer to "Introduction: Administrative commands" on page 3.
- **Programming**

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

  For more information, refer to "Introduction: Administrative programs" on page 3.
- **Data**

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

  For more information, refer to "Introduction: Administrative configuration data" on page 3.

Introduction: Administrative console

The administrative console is a graphical interface for performing deployment and system administration tasks. It runs in your Web browser. Your actions in the console modify a set of XML configuration files.

You can use the console to perform tasks such as:

- Add, delete, start, and stop application servers

- Deploy new applications to a server
- Start and stop existing applications, and modify certain configurations
- Add and delete Java 2 Platform, Enterprise Edition (J2EE) resource providers for applications that require data access, mail, URLs, and so on
- Manage variables, shared libraries, and other configurations that can span multiple application servers
- Configure product security, including access to the administrative console
- Collect data for performance and troubleshooting purposes
- Find the product version information. It is located on the front page of the console.

Starting and logging off the administrative console helps you begin using the console so that you can explore the available options. See also the **Reference > Administrator > Settings** section of the information center navigation. It lists the settings or properties you can configure.

Introduction: Administrative scripting (wsadmin)

The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language. The wsadmin tool is intended for production environments and unattended operations. You can use the wsadmin tool to perform the same tasks that you can perform using the administrative console.

The following list highlights the topics and tasks available with scripting:

- Getting started with scripting Provides an introduction to WebSphere Application Server scripting and information about using the wsadmin tool. Topics include information about the scripting languages and the scripting objects, and instructions for starting the wsadmin tool.
- Deploying applications Provides instructions for deploying and uninstalling applications. For example, stand-alone Java archive files and Web archive files, the administrative console, remote enterprise archive (EAR) files, file transfer applications, and so on.
- Managing deployed applications Includes tasks that you perform after the application is deployed. For example, starting and stopping applications, checking status, modifying listener address ports, querying application state, configuring a shared library, and so on.
- Configuring servers Provides instructions for configuring servers, such as creating a server, modifying and restarting the server, configuring the Java virtual machine, disabling a component, disabling a service, and so on.
- Configuring connections to Web servers Includes topics such as regenerating the plug-in, creating new virtual host templates, modifying virtual hosts, and so on.
- Managing servers Includes tasks that you use to manage servers. For example, stopping nodes, starting and stopping servers, querying a server state, starting a listener port, and so on.
- Configuring security Includes security tasks, for example, enabling and disabling administrative security, enabling and disabling Java 2 security, and so on.
- Configuring data access Includes topics such as configuring a Java DataBase Connectivity (JDBC) provider, defining a data source, configuring connection pools, and so on.
- Configuring messaging Includes topics about messaging, such as Java Message Service (JMS) connection, JMS provider, WebSphere queue connection factory, MQ topics, and so on.
- Configuring mail, URLs, and resource environment entries Includes topics such as mail providers, mail sessions, protocols, resource environment providers, referenceables, URL providers, URLs, and so on.
- Troubleshooting Provides information about how to troubleshoot using scripting. For example, tracing, thread dumps, profiles, and so on.
- Obtaining product information Includes tasks such as querying the product identification.
- Scripting reference material Includes all of the reference material related to scripting. Topics include the syntax for the wsadmin tool and for the administrative command framework, explanations and examples for all of the scripting object commands, the scripting properties, and so on.

Introduction: Administrative commands

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

See **Reference > Administrator > Commands** in the information center navigation area for the names and syntax of all the commands that are available with the product. A subset of these commands is particular to system administration purposes.

Introduction: Administrative programs

The product supports a Java programming interface for developing administrative programs. All of the administrative tools supplied with the product are written according to the API, which is based on the industry standard Java Management Extensions (JMX) specification. You can write a Java program that performs any of the administrative features of the WebSphere Application Server administrative tools. You can also extend the basic WebSphere Application Server administrative system to include your own managed resources.

Introduction: Administrative configuration data

The WebSphere Application Server product includes an implementation of the Java Management Extension (JMX) specification. All operations on managed resources in the product go through JMX functions. This setup means a more standard framework underlying your administrative operations as well as the ability to tap into the systems management infrastructure programmatically.

Introduction: Servers

Application servers

Application servers provide the core functionality of the WebSphere Application Server product family. They extend the ability of a Web server to handle Web application requests, and much more. An application server enables a server to generate a dynamic, customized response to a client request.

For additional overview, refer to “Introduction: Application servers.”

Introduction: Application servers

Overview

An application server is a Java Virtual Machine (JVM) that is running user applications. The application server collaborates with the Web server to return a dynamic, customized response to a client request. Application code, including servlets, JavaServer Pages (JSP) files, enterprise beans and their supporting classes, runs in an application server. Conforming to the Java 2 platform, Enterprise Edition (J2EE) component architecture, servlets and JSP files run in a Web container, and enterprise beans run in an Enterprise JavaBeans (EJB) container.

To begin creating and managing an application server, see Administering application servers.

You can define multiple application servers, each running its own JVM. Enhance the operation of an application server by using the following options:

- Configure transport chains to provide networking services to such functions as the service integration bus component of IBM service integration technologies, WebSphere Secure Caching Proxy, and the high availability manager core group bridge service. See Configuring transport chains for more information.
- Add an interface to an application server to define a hook point that runs when the server starts and shuts down. See Developing custom services for more information.

- Define command-line information that passes to a server when it starts or initializes. See startServer command for more information.
- Tuning application servers
- Enhance the performance of the application server JVM. See Configuring the JVM for more information.
- Use an Object Request Broker (ORB) for RMI/IIOP communication. See “Managing Object Request Brokers” on page 1807 for more information.

Asynchronous messaging

The product supports asynchronous messaging based on the Java Message Service (JMS) of a JMS provider that conforms to the JMS specification version 1.1.

The JMS functions of the default message service in WebSphere Application Server are served by one or more messaging engines (in a service integration bus) that runs within application servers.

Generic Servers

A generic server is a server that is managed in the WebSphere administrative domain, although it is not a server that is supplied by the WebSphere Application Server product. The generic server can be any server or process that is necessary to support the Application Server environment.

For more information, refer to Creating generic servers.

Related tasks

Creating generic servers
A generic server is a server that is managed in the WebSphere administrative domain, although it is not a server that is supplied by WebSphere Application Server. The WebSphere Application Server generic servers function enables you to define a generic server as an application server instance within the WebSphere Application Server administration, and associate it with a non-WebSphere server or process.

Programming to use JMS and messaging directly
Use these tasks to implement WebSphere J2EE applications that use JMS programming interfaces directly.

Administering application servers
An application server configuration provides settings that control how an application server provides services for running applications and their components.

Configuring transport chains
A transport chain consists of one or more types of channels, each of which supports a different type of I/O protocol, such as TCP or HTTP. Network ports can be shared among all of the channels within a chain. The channel framework function automatically distributes a request arriving on that port to the correct I/O protocol channel for processing.

Developing custom services
A custom service provides the ability to plug into a WebSphere Application Server application server to define a hook point that runs when the server starts and shuts down. An application server configuration provides settings that control how an application server provides services for running applications and their components.

Tuning application servers
The WebSphere Application Server contains interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application.

“Managing Object Request Brokers” on page 1807
Use this task to manage Object Request Brokers (ORB). An ORB manages the interaction between clients and servers using the Internet InterORB Protocol (IIOP).

Introduction: Web servers

In the WebSphere Application Server product, an application server works with a Web server to handle requests for dynamic content, such as servlets, from Web applications. SeeSupported Hardware and Software for this product for the most current information about supported Web servers.

The application server and Web server communicate using Web server plug-ins. Communicating with Web servers describes how to set up your Web server and Web server plug-in environment and how to create a Web server definition. The Web server definition associates a Web server with an application server. After you create a Web server definition, you can use the administrative console to perform the following functions for that Web server:

- Check the status of the Web server
- Generate a plug-in configuration file for that Web server.

If the Web server is an IBM HTTP Server and the IBM HTTP Server Administration server is installed and properly configured, you can also:

- Display the IBM HTTP Server Error log (error.log) and Access log (access.log) files.
- Start and stop the server.
- Display and edit the IBM HTTP Server configuration file (httpd.conf).
- Propagate the plug-in configuration file after it is generated.

You can not propagate a plug-in configuration file for a non-IBM HTTP Server. You must manually install an updated plug-in configuration file on that Web server.

After you set up your Web server and Web server plug-in, whenever you deploy a Web application, you must specify a Web server as the deployment target that serves as a router for requests to the Web application. The configuration settings in the plug-in configuration file (plugin-cfg.xml) for each Web server are based on the applications that are routed through that Web server. If the Web server plug-in configuration service is enabled, a Web server plug-in's configuration file is automatically regenerated whenever a new application is associated with that Web server.

Note: Before starting the Web server, make sure you are authorized to run any Application Response Measurement (ARM) agent associated with that Web server.

Refer to your Web server documentation for information on how to administer that Web server. For tips on tuning your Web server plug-in, see Web server plug-in tuning tips.

Introduction: Environment

Your WebSphere Application Server product environment includes Web server plug-ins, WebSphere Application Server variables, and other data objects. Configure values for settings in these categories using the Environment section of the administrative console.

Web servers

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

For more information, refer to "Introduction: Web servers."

Variables

A variable is a configuration property that can be used to provide a parameter for any value in the system. A variable has a name and a value to use in place of that name wherever the variable name is located within the system.

For more information, refer to “Variables.”

Related concepts

“Introduction: Web servers” on page 5

“Variables”
Variables in the WebSphere Application Server environment come in many varieties. They are used to control settings and properties relating to the server environment. Three main variable options that are important for a WebSphere Application Server user to know and understand are environment variables, and WebSphere Application Server variables, and custom properties.

Variables

Variables in the WebSphere Application Server environment come in many varieties. They are used to control settings and properties relating to the server environment. Three main variable options that are important for a WebSphere Application Server user to know and understand are environment variables, and WebSphere Application Server variables, and custom properties.

Environment variables. Environment variables, also called *native environment variables*, are not specific to WebSphere Application Server and are defined by other elements, such as UNIX, Language Environment (LE), or third-party vendors, among others. Some of the UNIX-specific native variables are LIBPATH and STEPLIB. These variables tend to be operating system-specific.

WebSphere Application Server variables

WebSphere Application Server variables are used for three purposes:

- Configuring WebSphere Application Server path names, such as JAVA_HOME, and APP_INSTALL_ROOT
- Configuring certain cell-wide customization values

WebSphere Application Server variables are specified in the administrative console by clicking **Environment > Manage WebSphere variables**. How the variable is set determines its scope.

A variable can apply to a node or a server.

If the variable is set:

- At the server level, it applies to the entire server.
- At the node level, it applies to all servers in the node, unless you set the same variable at the server level. In that case, for that server, the setting that is specified at the server level overrides the setting that is specified at the node level.

Custom properties

Custom properties are property settings meant for a specific functional component. Any configuration element can have a custom property. Common configuration elements are cell, node, server, Web container, and transaction service. A limited number of supported custom properties are available and these properties can be set in the administrative console using the custom properties link that is associated with the functional component.

For example, to set Web container custom properties, click **Servers > Application Servers >** *server_name* **> Web Container Settings > Web container > Custom Properties**

Custom properties set from the Web container custom properties page apply to all transports that are associated with that Web container; custom properties set from one of the Web container transport chain or HTTP transport custom properties pages apply only to that specific HTTP transport chain or HTTP transport. If the same property is set on both the Web container page and either a transport chain or HTTP transport page, the settings on the transport chain or HTTP transport page override the settings that are defined for the Web container for that specific transport.

Language versions offered by this product

This product is offered in several languages, as enabled by the operating platform on which the product is installed.

The following language versions are available:

- Brazilian Portuguese
- Chinese (Simplified)
- Chinese (Traditional)
- English
- French
- German
- Italian
- Japanese
- Korean
- Spanish

Chapter 2. How do I administer applications and their environments?

Hold your cursor over the task icon () to see a description of the task. The task preview feature is unavailable for Mozilla Web browsers.

See also the overview:

- Version 6 topology and terminology

Create WebSphere profiles	Documentation	
Administer configurations	Documentation	
Administer application servers	Documentation: • Console • Scripting - configure • Scripting - administer	
Administer other server types	Documentation: • Generic servers • Custom services	Guide me (Web servers)
Administer the UDDI registry	Documentation: • Configure • Administer	
Administer communication with Web servers (plug-ins)	Documentation: • Console • Scripting	Guide me
Administer HTTP sessions	Documentation: • Console • Scripting	
Administer IBM HTTP Server Version 6.x		
Provide access to naming and directory resources (JNDI)	Documentation: • Name server • Bindings	
Provide access to relational databases (JDBC resources)	Documentation: • Console • Scripting	Guide me
Provide access to messaging resources (default messaging provider)	Documentation: • Console • Scripting	
Use IBM service integration technologies		

Access Service Integration (SI) bus resources		Show me
Install applications	Documentation • Console • Scripting	
Start and stop applications	Documentation: • Console • Scripting	
Update applications	Documentation: • Console • Scripting	
Deploy and administer Web services applications	Documentation	
Choose an administrative client	Documentation	
Use the administrative console	Documentation	
Use scripting (wsadmin)	Documentation	
See also: • Start, stop, monitor processes • Other administrative commands		
Troubleshoot deployment	Documentation	
Troubleshoot administration	Documentation	

Legend for "How do I?..." links

Detailed steps	Show me	Tell me	Guide me	Teach me
Refer to the detailed steps and reference	Watch a brief multimedia demonstration	View the presentation for an overview	Be led through the console pages	Perform the tutorial with sample code
Approximate time: Varies	**Approximate time:** 3 to 5 minutes	**Approximate time:** 10 minutes+	**Approximate time:** 1/2 hour+	**Approximate time:** 1 hour+

Chapter 3. Starting and stopping quick reference

Start and stop servers in your application serving environment, referring to this quick guide to the administrative clients and several other tools that are provided with this product.

- Use commands to start and stop servers.

Quick reference: Issuing commands to start and stop servers
These examples are for starting and stopping the default profile on a server. Otherwise, you might need to specify `-profileName` *profile_name* when invoking the command.
Application server Run the following command. See startServer command for details and variations `startServer` *server* where *server* is the application server that you want to start.
Stopping the servers Use the same command as to start, except substitute stop for start. For example, to stop an application server, issue the command: `stopServer` *server*

- Use administrative clients and tools.

Quick reference: Opening the administrative console
To open the console using the default port, enter this Web address in your Web browser: `http://`*your_fully_qualified_server_name*`:9060/ibm/console` Depending on your configuration, your Web address might differ. Other factors can affect your ability to access the console. See Starting and logging off the administrative console for details, as needed.

  - To launch a scripting client, see Starting the wsadmin scripting client.
  - To learn about all available administrative clients, see Using the administrative clients.
  - For performance monitoring, see “Monitoring performance with Tivoli Performance Viewer (TPV)” on page 2258.

  See the administrator commands that are listed in the **Reference** section of the information center.
- Use development and deployment tools. The following tools can be used to edit deployment descriptors. A deployment descriptor is an Extensible Markup Language (XML) file that describes how to deploy a module or application by specifying configuration and container options.

  Assembly tools
  The assembly tools and Rational Web Developer provide a graphical interface for developing code artifacts, assembling the code artifacts into various archives (modules), and configuring related Java 2 Platform, Enterprise Edition (J2EE) Version 1.2, 1.3 or 1.4-compliant deployment descriptors.

  See Assembly tools for your tool options.

  Application Client Resource Configuration Tool (ACRCT)
  Use the ACRCT to configure deployment descriptors for client applications. See “Deploying J2EE application clients on workstation platforms” on page 289.

  Text editor
  It is not recommended because it has no built-in error checking or validation, but you can edit deployment descriptors with any text editor with which you can edit XML files.

- Use installation and customization tools. Use the following tools to find planning information, start the product installation, and perform customization and other activities after installation.

  Launchpad
  A graphical interface for launching the product installation. It also provides links to information you might need for installation.

  See Using the launchpad to start the installation.

  First Steps
  A desktop GUI from which you can start or stop the Application Server. It also provides access to the administrative console.

  See firststeps command.

  Profile Management tool
  The installation places the product binary files on a machine and creates a default profile.
- Use troubleshooting tools.

  See Working with troubleshooting tools.

Chapter 4. Backing up and recovering the application serving environment

The product uses many operating system and application resources that you should consider adding to your backup and recovery procedures.

WebSphere Application Server resources can be saved while the product environment is active. When backing up database data, you may have to shut down some or all services if a snapshot cannot be obtained. This would occur if there are requests which obtain locks or have open transactions against the database being saved. In a distributed environment, you may need to consider how to get a consistent backup across several systems. If the data on systems is not closely related to data on other systems, you may be able to backup each system in isolation. If you need a snapshot across systems simultaneously, you may need to stop activity on all systems while the snapshot is taken.

How often you back up resources depends largely on when or how often you expect them to change.

- Back up your product environment configuration.

  This category covers the resources that define your WebSphere Application Server operating environment. Once you have done initial setup, this information should change very infrequently. You might backup this information only when you change these settings, and not include these resources in regularly scheduled backups.

  – Administrative configuration files
  – HTTP configuration (see the documentation for your Web server)
- Back up your applications.

  This category covers the applications you run using WebSphere Application Server. You should back these up the same way you back up other applications on your system. You could backup these resources every time you add or change an application, or include these resources in a regularly scheduled backup.

  – Application deployment configuration files
- Back up your application data.

  This category covers the data stores used by your WebSphere Application Server applications. Unless your applications serve only static information, these resources are usually quite dynamic. You should back these up the same way you back up other business data on your system. These resources are suited for inclusion in a regularly scheduled backup.

  – Servlet session data

If your applications are using other resources or services that are external to the product, remember to include those in your backup plan as well.

Chapter 5. 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.

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 24 refers to additional sources.

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.

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 a shared library instance for each library file that your applications need.
   b. Define a library reference instance for each shared library that your application uses.

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”
- “Class-loader isolation policies” on page 17
- “Class-loader modes” on page 19

Class loaders used and the order of use

The runtime environment of WebSphere Application Server 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.

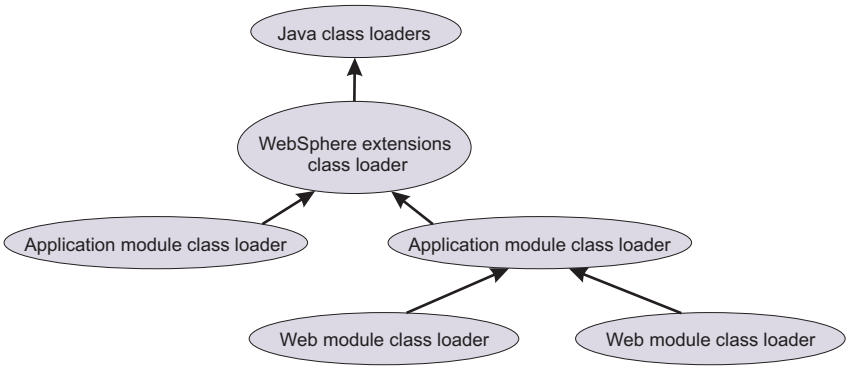
   The CLASSPATH class loader loads the Java 2 Platform, Enterprise Edition (J2EE) application programming interfaces (APIs) provided by the WebSphere Application Server product in the `j2ee.jar` file. Because this class loader loads the J2EE APIs, you can add libraries that depend on the J2EE APIs to the 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 ZIP 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 J2EE class-loading rules to load classes and JAR files from an enterprise application. WebSphere Application Server 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 J2EE 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:

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 (Single) or unique for each application (Multiple). The application class-loader policy controls the isolation of applications that are running in the system. When set to `Single`, applications are not isolated. When set to `Multiple`, applications are isolated from each other.
WAR	By default, Web module class loaders load the contents of the `WEB-INF/classes` and `WEB-INF/lib` 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. The WAR class-loader policy controls the isolation of Web modules. If this policy is set to `Application`, 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 `Module`, then each Web module receives its own class loader whose parent is the application class loader. **Tip:** The console and the underlying `deployment.xml` file use different names for WAR class-loader policy values. In the console, the WAR class-loader policy values are `Application` or `Module`. However, in the underlying `deployment.xml` file where the policy is set, the WAR class-loader policy values are `Single` instead of `Application`, or `Multiple` instead of `Module`. `Application` is the same as `Single`, and `Module` is the same as `Multiple`.

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

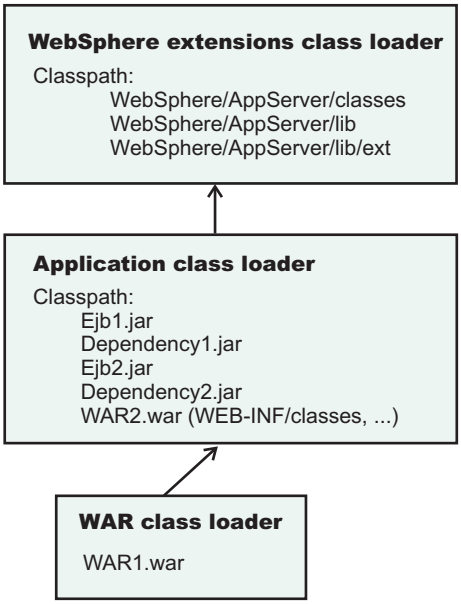
For each application server in the system, you can set the application class-loader policy to `Single` or `Multiple`. When the application class-loader policy is set to `Single`, then a single application class loader loads all EJB modules, dependency JAR files, and shared libraries in the system. When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

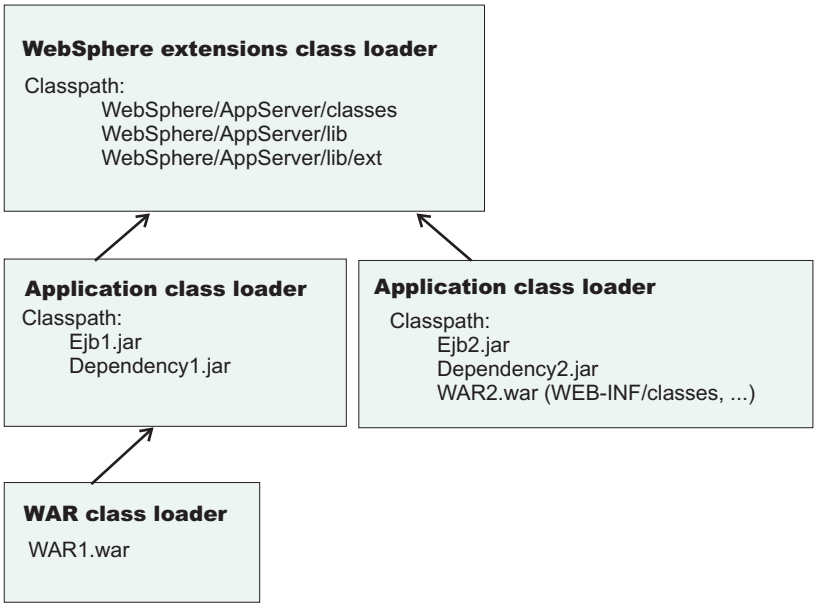
```
Server's application class-loader policy: Single
Application's WAR class-loader policy: Module

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:

Class-loader mode	Description
Parent first Also known as **Classes loaded with parent class loader first**.	The `Parent first` or `Classes loaded with parent class loader 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 application class loader first**.	The `Parent last` or `Classes loaded with application class loader first` 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.

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

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.

1. Click **Servers > Application Servers >** *server_name* to access the settings page for an application server.
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 settings page for the application server 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 settings page for the application server, select either of the following values:

Option	Description
Parent 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. `Parent first` is the default value for class loading mode.
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 settings page for the application server, 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 settings page for a class loader, 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 application class loader first` 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.

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 > Application servers >** *server_name* **> Java and Process Management > Class loader**.

Class loader ID

Provides 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`. By specifying `Classes loaded with application class loader first`, 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 > Application servers >** *server_name* **> Java and Process Management > Class loader >** *class_loader_ID*.

Class loader ID

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

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 application class loader first`, 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 application class loader first`. 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 WebSphere Application Server, 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 in WebSphere Application Server, 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 application class loader first`. 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.

Data type String

Default Classes loaded with parent class loader 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.

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

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 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.

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.

1. Click **Applications > Enterprise Applications >** *application_name* **> Class loading and update detection** to access the settings page for an application class loader.
2. Specify whether to reload application classes when the application or its files are updated.

   By default, class reloading is not enabled. Select **Reload classes when application files are updated** 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.
3. 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.xmi`) 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).
4. 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 **Class 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.

Option	Description
Classes loaded with application class loader first	Causes the class loader to search in the application class loader first to load a class. By specifying `Classes loaded with application class loader first`, your application can override classes contained in the parent class loader. **Attention:** Specifying the `Classes loaded with application class loader first` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

5. 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**.

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.

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

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` on the settings page of an application class loader, then the application class loader loads the Web module contents as well as the enterprise bean (EJB) modules, shared libraries, resource adapter archives (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.

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

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

Settings such as **Reload classes when application files are updated**, **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	Causes the class loader to search in the parent application class loader first to load a class. This is the standard for Development Kit class loaders and WebSphere Application Server class loaders. **Tip:** If classes and resources needed by the Web module cannot be accessed by the application class loader, but can be accessed by the WAR class loader, specify `Classes loaded with application class loader first`. If the application class loader cannot find a class, the class loader delegates the request to find the class to its parent, the WebSphere Application Server extensions class loader. If the WebSphere Application Server extensions class loader cannot find the class, the class loader delegates the request to its parent, the bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine. Requests can only go to a parent class loader; they cannot go to a child class loader. Thus, if `Classes loaded with parent class loader first` is specified, the WAR class loader never receives a request to load a class.
Classes loaded with application class loader first	Causes the class loader to search in the WAR class loader first to load a class. By specifying `Classes loaded with application class loader first`, your WAR class loader can override classes contained in the parent application class loader. **Attention:** Specifying the `Classes loaded with application class loader first` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

3. Click **OK**.

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 that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Refer to Web resources for learning for links to information applicable to WebSphere Application Server generally, such as lists of IBM technical papers, Redbooks and samples.

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 that is needed to resolve this problem. Before opening a PMR, see the IBM Support page.

View links to additional information about:
- "Programming model and decisions"
- "Programming instructions and examples"
- "Programming specifications"

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)
- 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)
- Demystifying class loading problems, Part 3: Tackling more unusual class loading problems - Understand class loading and quash subtle exceptions (*developerWorks*, December 2005)
- J2EE Class Loading Demystified (*developerWorks*, August 2002)
- 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)

Programming instructions and examples
- WebSphere Application Server V6 System Management & Configuration Handbook
- *IBM WebSphere Developer Technical Journal:* Co-hosting multiple versions of J2EE applications
- Chapter 24 - J2EE Packaging and Deployment excerpted from Professional Java Server Programming J2EE 1.3 Edition

Programming specifications
- J2EE™ Platform Specification
- J2EE™ Extension Mechanism Architecture

Chapter 6. Deploying and administering applications

Deploying an application file consists of installing the application file on a server configured to hold installable modules.

Before installing an enterprise application or other installable module on an application server, you must develop the module, assemble the module, and configure the target server . Before choosing a deployment target for the module, ensure that the target version is compatible with your module.

During installation, you can configure the module enough to enable it to run on the server. After installation, you can configure the module further, start or stop the application, and otherwise manage its activity.

The topics in this section describe how to deploy and administer applications or modules using the administrative console. You can also use scripting or administrative programs (JMX).

- Install application files on an application server.
- Edit the administrative configuration for an application.
- **Optional:** View the deployment descriptor for an application or module.
- Start and stop the application.
- Export applications.
- Export DDL files.
- Update an application or module.
- Uninstall applications.
- Remove a file from an application or module.

After making changes to administrative configurations of your applications in the administrative console, ensure that you save the changes.

Enterprise (J2EE) applications

Enterprise applications (or J2EE applications) are applications that conform to the Java 2 Platform, Enterprise Edition, specification.

Enterprise applications can consist of the following:

- Zero or more EJB modules (packaged in JAR files)
- Zero or more Web modules (packaged in WAR files)
- Zero or more connector modules (packaged in RAR files)
- Zero or more Session Initiation Protocol (SIP) modules (packaged in SAR files)
- Zero or more application client modules
- Additional JAR files containing dependent classes or other components required by the application
- Any combination of the above

A J2EE application is represented by, and packaged in, an enterprise archive (EAR) file.

System applications

A *system application* is a J2EE enterprise application that is central to a WebSphere Application Server product.

Examples of system applications include *isclite*, *managementEJB* and *filetransfer*.

Because a system application is an important part of a WebSphere Application Server product, a system application is deployed when the product is installed and is updated only through a product fix or upgrade. For some system applications, such as *filetransfer*, users cannot change the metadata for the system application, unless the metadata assigns users and groups for security purposes. For these applications, non-security related metadata such as its J2EE bindings or J2EE extensions must be updated through a product fix or upgrade.

System applications are not shown in the list of installed applications on the console Enterprise Applications page, or through wsadmin and Java application programming interfaces, to prevent users from accidentally stopping, updating or removing the system applications.

Note that J2EE Samples are not system applications even though they are provided as part of a WebSphere Application Server product. Similarly, applications that support changes to their metadata are not system applications.

Installing application files

As part of deploying an application, you install application files on a server configured to hold installable modules.

Before you can install your application files on an application server, you must configure the target application server. As part of configuring the server, determine whether your application files can be installed to your deployment targets.

Also, before you install the files, assemble modules as needed.

Installable modules include enterprise archive (EAR), enterprise bean (EJB), Web archive (WAR), Session Initiation Protocol (SIP) module (SAR), resource adapter (connector or RAR), and 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.

1. Determine which method to use to install your application files. WebSphere Application Server provides several ways to install modules.
2. Install the application files using
   - Administrative console
   - wsadmin scripts
   - Java administrative programs that use JMX APIs
   - Java programs that define a J2EE DeploymentManager object in accordance with J2EE Deployment API 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 J2EE DeploymentManager object in accordance with J2EE Deployment API Specification (JSR-88)

Save the changes to your administrative configuration.

Next, test the application. For example, point a Web browser at the URL for a deployed application (typically http://*hostname*:9060/*Web_module_name*, where *hostname* is your valid Web server and 9060 is the default port number) and examine the performance of the application. If the application does not perform as desired, edit the application configuration, then save and test it again.

Installable module versions

The contents of a module affect whether you can install the module on a WebSphere Application Server Version 6.0 and later (6.x) deployment target, or if you must install the module on a Version 5.0 and later (5.x) deployment target.

Installable application modules

You can install an application, enterprise bean (EJB) module, Session Initiation Protocol (SIP) module (SAR), or Web module developed for a Version 5.x product on a 5.x or 6.x deployment target, provided the module:

- Does not support Java 2 Platform, Enterprise Edition (J2EE) 1.4;
- Does not call any 6.x runtime application programming interfaces (APIs); and
- Does not use any 6.x product features.

If the module supports J2EE 1.4, then you must install the module on a 6.x deployment target. If the module calls a 6.1.x API or uses a 6.1.x feature, then you must install the module on a 6.1.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 or 6.1.x deployment target.

Selecting options such as **Precompile JavaServer Pages files**, **Use binary configuration**, **Deploy Web services** or **Deploy enterprise beans** during application installation indicates that the application uses 6.1.x product features. You cannot deploy such applications on a 5.x or 6.0.x deployment target. You must deploy such applications on a 6.1.x deployment target.

Similarly, you must deploy an application that uses J2EE 1.4 features such as Java Authorization Contract for Containers (JACC) provided by an application server on a 6.x deployment target.

Installable RAR files

You can install a standalone resource adapter (connector) module, or RAR file, developed for a Version 5.x product to a 5.x or 6.x deployment target, provided the module does not call any 6.x runtime APIs. If the module calls a 6.x API, then you must install the module on a 6.x deployment target.

Deployment targets

A *5.x deployment target* is a server on a WebSphere Application Server Version 5 product.

A *6.x deployment target* is a server on a WebSphere Application Server Version 6 product.

Table 1. Compatible deployment target versions for 5.x and 6.x modules

Module type	Module Java support	Module calls 6.x runtime APIs or uses 6.x features?	Client versions that can install module	Deployment target versions
Application, EJB, Web, or client	J2EE 1.3	No	5.x or 6.x	5.x or 6.x

Table 1. Compatible deployment target versions for 5.x and 6.x modules (continued)

Application, EJB, Web, or client	J2EE 1.3	Yes	6.x	6.x Modules that call 6.1.x runtime APIs or use 6.1.x features must be installed on a 6.1.x deployment target. Modules that call 6.0.x runtime APIs or use 6.0.x features can be installed on any 6.x deployment target.
Application, EJB, SAR, Web, or client	J2EE 1.4	Yes or No	6.x	6.x
Resource adapter	JCA 1.0	No	5.x or 6.x	5.x or 6.x
Resource adapter	JCA 1.0	Yes	6.x	6.x Modules that call 6.1.x runtime APIs must be installed on a 6.1.x deployment target. Modules that call 6.0.x runtime APIs can be installed on any 6.x deployment target.
Resource adapter	JCA 1.5	Yes or No	6.x	6.x Modules that call 6.1.x runtime APIs must be installed on a 6.1.x deployment target. Modules that call 6.0.x runtime APIs can be installed on any 6.x deployment target.

Ways to install applications or modules

The product provides several ways to install application files.

Installable files include enterprise archive (EAR), enterprise bean (EJB), Web archive (WAR), Session Initiation Protocol (SIP) module (SAR), resource adapter (connector or RAR), and application client modules. They can be installed on a server. Application client files can be installed in a WebSphere Application Server configuration but cannot be run on a server.

Table 2. Ways to install application files

Option	Method	Modules	Comments	Starting after install
Administrative console install wizard See “Installing application files with the console” on page 32.	Click **Applications > Install New Application** in the console navigation tree and follow instructions in the wizard.	All EAR, EJB, WAR, SAR, RAR, and application client files	Provides one of the easier ways to install application files. See “Preparing for application installation settings” on page 37 for guidance. For applications that do not require changes to the default bindings, select **Show me all installation options and parameters**, select **Generate default bindings**, click the Summary step, and then click **Finish**.	Click **Start** on the Enterprise Applications page accessed by clicking **Applications > Enterprise Applications** in the console navigation tree.
wsadmin scripts	Invoke AdminApp object *install* commands in a script or at a command prompt.	All EAR, EJB, WAR, SAR, RAR, and application client files	Getting started with scripting provides an overview of wsadmin.	• 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 MBeans to install the application. Managing applications through programming provides an overview of Java MBean programming.	Start the application by calling the *startApplication* method on a proxy.
WebSphere rapid deployment Refer to articles under Rapid deployment of J2EE applications in this information center.	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.	All J2EE modules, including EAR files and standalone EJB, WAR, SAR, RAR, and application client files	WebSphere rapid deployment offers the following advantages: • 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.	Use any of the above options to start the application. Clicking **Start** on the Enterprise Applications page is the easiest option.

Table 2. Ways to install application files (continued)

Option	Method	Modules	Comments	Starting after install
Java programs	Code programs that use J2EE DeploymentManager (JSR-88) methods.	All J2EE modules, including EAR files and standalone EJB, WAR, SAR, RAR, and application client files	• Uses J2EE Application Deployment Specification (JSR-88). • Can customize modules using DConfigBeans.	Call the J2EE DeploymentManager (JSR-88) method *start* in a program to start the deployed modules when the module's running environment initializes.

Installing application files with the console

Installing 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 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.

To install new enterprise application files to a WebSphere Application Server configuration, you can use the administrative console, the wsadmin tool, Java MBean programs, or Java programs that call J2EE DeploymentManager (JSR-88) methods. This topic describes how to use the administrative console to install an application, EJB component, Session Initiation Protocol (SIP) module (SAR), or Web module.

Important: After you start performing the steps below, 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.

1. Click **Applications > Install New 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 standalone Web application archive (WAR), SAR or Java archive (JAR) file for installation.
   b. If you are installing a standalone WAR or SAR file, specify the context root.
   c. Select whether to view all installation options.
      Prompt me 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.
      Show me all installation options and parameters
      Displays all installation options. To use **Generate default bindings**, which supplies default values for incomplete bindings, select this option.
   d. Click **Next**.
3. If you selected **Show me all installation options and parameters**, for the second Preparing for application installation page:
   a. Select whether to generate default bindings. Using the default bindings causes any incomplete bindings in the application to be filled in with default values. Existing bindings are not altered. You can customize default values used in generating default bindings. For example, you can specify a Java Naming and Directory Interface (JNDI) prefix for EJB files in EJB modules, default data

source and connection factory settings for EJB modules, virtual host for Web modules, and so on. “Preparing for application installation settings” on page 37 describes available customizations and provides sample bindings.

b. 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 51 provides sample steps.

4. Specify values for installation options as needed.

   You can click on a step number to move directly to that panel instead of clicking **Next**.

Panel	Description
Select installation options	On the **Select installation options** panel, provide values for the settings specific to WebSphere Application Server. Default values are used if you do not specify a value.
Map modules to servers	On the **Map modules to servers** panel, 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. A deployment target can be an application server or Web server.
Provide options to compile JSPs	If the **Precompile JavaServer Pages files** setting is enabled on the **Select installation options** panel and your application uses JavaServer Pages (JSP) files, then you can specify JSP compiler options on the **Provide options to compile JSPs** panel.
Provide JNDI names for beans	If your application uses EJB modules, on the **Provide JNDI names for beans** panel, specify a JNDI name for each enterprise bean in every EJB module. You must specify a JNDI name for every enterprise bean defined in the application. For example, for the EJB module `MyBean.jar`, specify `MyBean`.
Map default data sources for modules containing 1.x entity beans	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.
Map data sources for all 1.x CMP beans	If your application has CMP beans that are based on the EJB 1.x specification, for **Map data sources for all 1.x CMP beans**, specify a JNDI name for data sources to be used for each of the 1.x CMP beans. 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 neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error displays after you click **Finish** and the installation is cancelled.
Map EJB references to beans	If your application defines EJB references, for Map EJB references to beans, specify JNDI names for enterprise beans that represent the logical names specified in EJB references. Each EJB reference defined in the application must be bound to an EJB file before clicking **Finish** on the Summary panel.
Map resource references to resources	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. Each resource reference defined in the application must be bound to a resource defined in your WebSphere Application Server configuration before clicking on **Finish** on the Summary panel.

Panel	Description
Map virtual hosts for Web modules	If your application uses Web modules, for **Map virtual hosts for Web modules**, select a virtual host from the list that should 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 panel.
Map security roles to users or groups	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 of 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: 1. Select a role and click **Lookup users** or **Lookup groups**. 2. On the Lookup users or groups panel 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** panel.
Map RunAs roles to users	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 of the roles or select individual roles. After selecting a role, enter values for the user name, password, and verify password and click **Apply**.
Ensure all unprotected 1.x methods have the correct level of protection	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.
Bind listeners for message-driven beans	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.
Map default data sources for modules containing 2.x entity beans	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. The default data source for EJB modules is optional if data sources are specified for individual CMP beans.
Map data sources for all 2.x CMP beans	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** panel, for each of the 2.x CMP beans specify a JNDI name and the type of resource authorization for data sources to be used. 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 neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error is displayed after you click **Finish** and installation is cancelled.

Panel	Description
Ensure all unprotected 2.x methods have the correct level of protection	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** panel, specify whether you want to assign a specific role to the unprotected methods, add the methods to the exclude list, or mark them as unchecked. Methods added to the exclude list are marked as uncallable. For methods marked unchecked no authorization check is performed prior to their invocation.
Provide options to perform the EJB Deploy	If the **Deploy enterprise beans** setting is enabled on the **Select installation options** panel, then you can specify options for the EJB deployment tool on the **Provide options to perform the EJB Deploy** panel. On this panel, you can specify extra class paths, RMIC options, database types, and database schema names to be used while running the EJB deployment tool.
Map shared libraries	On the **Shared library references** and **Shared library mapping** panels, 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.
Provide JSP reloading options for Web modules	If your application uses Web modules, for **Provide JSP reloading options for Web modules**, configure the class reloading of JavaServer Pages (JSP) files.
Map context roots for Web modules	If your application uses Web modules, for **Map context roots for Web modules**, specify a context root for each Web module in the application.
Initialize parameters for servlets	If your application uses Web modules, for **Initialize parameters for servlets**, specify or override initial parameters that are passed to the init method of Web module servlet filters.
Map environment entries for Web modules	If your application uses Web modules, for **Map environment entries for Web modules**, configure the environment entries of Web modules such as servlets and JSP files.
Map resource environment entry references to resources	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.
Correct use of system identity	If your application defines **Run-As Identity** as *System Identity*, for **Correct use of system identity**, you can optionally change it to *Run-As role* and specify a user name and password for the Run As role specified. Selecting *System Identity* 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.
Bind message destination references to administered objects	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. **Attention:** If multiple message destination references are linked 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.
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 J2C connection factory, J2C administered object and J2C activation specification.

Panel	Description
Bind J2C activationspecs to destination JNDI names	If your application contains an embedded `.rar` file, its activationSpec property has the value `Destination`, and its introspected type is `javax.jms.Destination`, for **Bind J2C activationspecs to destination JNDI names**, specify the jndiName value for each activation bound to it.
Select current backend ID	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. For information on backend databases, refer to EJB deployment tool. This step is not shown if the **Deploy enterprise beans** setting is enabled on the **Select installation options** panel and if a database type other than `None` is specified on the **Provide options to perform the EJB Deploy** panel.
Provide options to perform the Web services deployment	If the **Deploy Web services** setting is enabled on the **Select installation options** panel and your application uses Web services, then you can specify `wsdeploy` command options on the **Provide options to perform the Web services deployment** panel. For information on this panel, refer to descriptions of the `wsdeploy -cp` and `-jardir` options.

5. On the Summary panel, 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 panel.
   d. Click **Finish**.

Several messages are displayed, indicating whether your application file is installing successfully.

If **Validate input off/warn/fail** on the **Select installation options** panel 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 exception 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 exception, 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 exception, check the options you specified on the Java Virtual Machine page of the application server running the administrative console. Then, try installing the application file again.

Windows During installation certain application files are extracted in the directory represented by the configuration session and, when the configuration is saved, these files are saved in the WebSphere Application Server configuration repository. On Windows machines, there is a limit of 256 characters for file paths. Therefore, the application installation might fail if the path for application files in the configuration session or in the configuration repository exceeds the limit of 256 characters. You might see FileNotFound exceptions with *path name too long* in the message. To overcome such problems, make application names and module URI names shorter in length, which helps reduce the file path length. Then, try installing the application file again.

After the application file installs successfully, do the following:

1. Save the changes to your configuration.

   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 installation, application files are copied to the destination directory when the changes are saved.

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.

Preparing for application installation settings

Use this page to install an application (EAR file) or module (JAR, SAR or WAR file).

To view this administrative console page, click **Applications > Install New Application**.

Follow the steps on this page to install an application or module. You must complete, at minimum, the first step; you must complete some or all of the later steps, depending on whether you are installing an application, EJB module, SIP module or Web module.

Path to the new application:

Specifies the fully qualified path to the `.ear`, `.jar`, `.sar`, or `.war` file for the enterprise application.

Use **Local file system** if the browser and application files are on the same machine (whether or not the server is on that machine, too).

Use **Remote file system** if the application file resides on any node in the current cell context. Only `.ear`, `.jar`, `.sar`, or `.war` files are shown during the browsing.

During application installation, application files typically are uploaded from a client machine running the browser to the server machine running the administrative console, where they are deployed. In such cases, use the Web browser running the administrative console to select EAR, WAR, SAR or JAR modules to upload to the server machine.

In some cases, however, the application files reside on the file system of any of the nodes in a cell. To have the application server install these files, use the **Remote file system** option.

Also use the **Remote file system** option to specify an application file already residing on the machine running the application server. For example, the field value might be *profile_root*`/installableApps/test.ear`. If you are installing a standalone WAR module, then specify the context root as well.

After the application file is transferred, the **Remote file system** value shows the path of the temporary location on the deployment manager or server machine.

Context root:

Specifies the context root of the Web application (WAR)or a Session Initiation Protocol (SIP) module (SAR).

This field is used only to install a standalone WAR or SAR file. 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`.

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.

The **Prompt me only when additional information is required** option enables you to install your application more easily because you do not need to examine all available installation options.

However, to use the **Generate default bindings** option, which might be the quickest and easiest option for installing your application, you must select the **Show me all installation options and parameters** and then select **Generate default bindings** on the next panel.

Generate default bindings:

Specifies whether to generate default bindings. If you place a check mark in the check box, then any incomplete bindings in the application are filled in with default values. Existing bindings are not altered.

By choosing this option, you can directly jump to the Summary step and install the application if none of the steps have a red asterisk (*) next to them. A red asterisk denotes that the step has incomplete data and requires a valid value. On the Summary panel, verify the cell, node and server on which the application is installed.

You must select **Show me all installation options and parameters** to view this option.

Bindings are generated as follows:
- EJB JNDI names are generated of 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.
- EJB references are bound as follows: If an `<ejb-link>` is found, it is honored. 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.
- Resource reference bindings are derived from the `<res-ref-name>` tag. Note that 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 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 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 password options. This results in default data source settings for each EJB JAR file. No bean-level data source bindings are generated.
- For EJB2.1 or EJB2.0 message-driven beans deployed as 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 as follows: if a `<message-destination-link>` is found then the JNDI name is set to ejs/*message-destination-link*Name. Otherwise the JNDI name is set to eis/*message-destination-ref*Name.
- For EJB 2.0 message-driven beans deployed against a listener ports, the listener ports are derived from the MDB `<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, it does not work if:
- You want to explicitly control the global JNDI names of one or more EJB files.
- You need tighter control of data source bindings for container-managed persistence (CMP) beans. That is, you have multiple data sources and need more than one global data source.
- You must 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 (a custom strategy). Use the **Specific bindings file** field to specify a custom strategy and see the field's help for examples.

Prefixes:

Specifies prefixes to use for generated JNDI names.

You must select **Show me all installation options and parameters** to view prefix options.

Override:

Specifies whether generated bindings are to override existing bindings.

If **Override existing bindings** is selected, the existing bindings are overridden by the generated ones.

You must select **Show me all installation options and parameters** to view override options.

EJB 1.1 CMP bindings:

Specifies the default data source JNDI name.

If the **Default bindings for EJB 1.1 CMPs** radio button is selected, specify the JNDI name for the default data source to be used with the container-managed persistence (CMP) 1.1 beans. Also specify the user ID and password for this default data source.

You must select **Show me all installation options and parameters** to view EJB CMP binding options.

Data source bindings for 2.0 CMP beans:

Specifies the default data source JNDI name for 2.0 CMP beans.

You must select **Show me all installation options and parameters** to view data source binding options.

Virtual host:

Specifies the virtual host for the Web module.

You must select **Show me all installation options and parameters** to view virtual host options.

Specific bindings file:

Specifies a bindings file that overrides the default binding.

You must select **Show me all installation options and parameters** to view this option.

Change the behavior of the default binding with an XML document (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.

Brief examples of how to override various aspects of the default bindings generator follow:

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>
```

Note: 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

Example 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/somAO</jndi-name>
       </message-destination-ref-binding>
```

```
      </message-destination-ref-bindings>
     </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>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 J2EE client JAR file

Example code for overriding a resource reference binding from a WAR file follows. Use similar code to override a resource reference binding from an enterprise bean (EJB) JAR file or a J2EE client JAR file.

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <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>
</dfltbndngs>
```

Restriction: Ensure that the setting for <resource-ref-name> matches the resource-ref tag in the deployment descriptor. Here 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

Example XML extract in a custom strategy file for overriding the JMS activationSpec JNDI name for an EJB 2.1 or EJB 2.0 message-driven bean deployed as a JCA 1.5-compliant resource.

```
<?xml version="1.0"?>
 <!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
  <dfltbndngs>
  <module-bindings>
   <ejb-jar-binding>
     <jar-name>YourEjbJar.jar</jar-name>
     <ejb-bindings>
      <ejb-binding>
        <ejb-name>YourMDB</ejb-name>
        <activationspec-jndi-name>activationSpecJNDI</activationspec-jndi-name>
      </ejb-binding>
     </ejb-bindings>
   </ejb-jar-binding>
  </module-bindings>
 </dfltbndngs>
```

Overriding the JMS listener port name for an EJB 2.0 message-driven bean

Example XML extract in a custom strategy file for overriding the JMS listener port name for an EJB 2.0 message-driven bean deployed against a listener port.

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <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>
</dfltbndngs>
```

Overriding an EJB reference binding from an EJB JAR, WAR file, or EJB file

Example code for overriding an EJB reference binding from an EJB JAR file follows. Use similar code to override an EJB reference binding from a WAR file or an EJB file.

```
<?xml version="1.0"?>
<!DOCTYPE dfltbndngs SYSTEM "dfltbndngs.dtd">
<dfltbndngs>
  <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>
</dfltbndngs>
```

Select installation options settings

Use this panel to specify options for the installation of an 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 panel, click **Applications > Install New Application** and then specify values as needed for your application on the Preparing for application installation pages. The Select installation options panel 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 6.1 deployment target.

If you select **Precompile JavaServer Pages files** and try installing your application onto an earlier deployment target such as version 5.x, the installation is rejected. You can deploy applications to only those targets that have same WebSphere version as the deployment manager. If applications are targeted to servers that have an earlier version than the deployment manager, then you cannot deploy to those targets.

Data type	Boolean
Default	False

Directory to install application:

Specifies the directory to which the enterprise application (EAR) file will be installed.

The default value is the value of APP_INSTALL_ROOT/*cell_name*, where the APP_INSTALL_ROOT variable is *app_server_root*/installedApps; for example, *app_server_root*/installedApps/*cell_name*.

You can specify an absolute path or use a pathmap variable such as ${MY_APPS}. You can use a pathmap variable in any installation.

This **Directory to install application** field is the same as the **Location (full path)** setting on an Application binaries page.

Data type	String
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. The binaries are deleted when you uninstall and save changes to the configuration, and, on the Network Deployment product, 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.

Important: 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.

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 application resource (EAR) file. Select this setting for applications installed on 6.x deployment targets only. This setting is not valid for applications installed on 5.x deployment targets.

This **Use binary configuration** field is the same as the **Use configuration information in binary** setting on an Application binaries page.

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 bean (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, Rational Web Developer or Application Server Toolkit (AST) and the EJBDeploy tool was not run during assembly.
- The EAR file was not assembled using an assembly tool such as Rational Application Developer, Rational Web Developer or AST.
- The EAR file was assembled using versions of the Application Assembly Tool (AAT) previous to Version 5.

For this option, install only onto a 6.1 deployment target.

If you select **Deploy enterprise beans** and try installing your application onto an earlier deployment target such as version 5.x, the installation is rejected. You can deploy applications to only those targets that have same WebSphere version as the deployment manager. If applications are targeted to servers that have an earlier version than the deployment manager, 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** panel, 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** panel.

The default database type is `DB2UDB_V81`.

Enabling this setting might cause the installation program to run for several minutes.

Data type Boolean
Default true

Application name:

Specifies a logical name for the application. An application name must be unique within a cell and cannot contain an unallowed character.

An application name cannot begin with a period (**.**), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unallowed 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.

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.

Data type	Boolean
Default	true

Enable class reloading:

Specifies whether the WebSphere Application Server 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 **Enable class reloading** field is the same as the **Reload classes when application files are updated** setting on an Class loading and update detection page.

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.xmi`) 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 an Class loading and update detection page.

Data type	Integer
Units	Seconds
Default	3

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 6.1 deployment target.

If you select **Deploy Web services** and try installing your application onto an earlier deployment target such as version 5.x, the installation is rejected. You can deploy applications to only those targets that have same WebSphere version as the deployment manager. If applications are targeted to servers that have an

earlier version than the deployment manager, then you cannot deploy to those targets.

Data type Boolean
Default false

Validate input off/warn/fail:

Specifies whether WebSphere Application Server 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.

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`.

Data type Boolean
Default false

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 drop-down list. Drop-down list selections overwrite file permissions set in the text field.

You can set one or more of the following file permission strings in the drop-down list. Selecting multiple options combines the file permission strings.

Drop-down 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`

Drop-down list option	File permission string set
Allow HTML and image files to be read by everyone	`.*\.htm=755#.*\.html=755#.*\.gif=755#.*\.jpg=755`

Instead of using the drop-down 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.

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 following directory and file URIs are processed during a file permission operation:

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.

Windows You cannot unset read permission on a file on Windows platforms. With POSIX style permission bits, the bit for denoting `readable` on a file is 4, `writable` is 2, and `executable` is 1. Thus, permission of a file on a Windows platform is either 5 or 7. Also, in POSIX style there are `user`, `group` and `world` permissions. You can only set the `user` permission for a file on Windows platforms. The `group` and `world` permission bits are ignored.

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.

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.

Data type String

Provide options to perform the EJB Deploy settings

Use this panel 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 panel is a step in the application installation and update wizards. To view this panel, you must select **Deploy enterprise beans** on the **Select installation options** panel. Thus, to view this panel, click **Applications > Install New Application >** *application_path* **> Show me all installation options and parameters > Next > Next > Deploy enterprise beans > Next > Step: Provide options to perform the EJB Deploy**.

You can specify the EJB deployment tool options on this panel only when installing or updating an application that contains EJB 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** panel of the wizard.

Deploy EJB option - Class path:

Specifies the class path of one or more zipped or Java archive (JAR) files on which the Java archive (JAR) or EAR file being installed depends.

To specify the class paths of multiple zipped and JAR files, the zipped and JAR file names must be fully qualified, separated by semicolons, and enclosed in double quotation marks. For example:

path`\myJar1.jar;`*path*`\myJar2.jar;`*path*`\myJar3.jar`

Deploy EJB option - Class path is the same as the `ejbdeploy` command parameter `-cp` *`class_path`*.

Data type	String
Default	null

Deploy EJB option - 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"
```

Deploy EJB option - RMIC is the same as the `ejbdeploy` command parameter `-rmic` *`"options"`*.

Data type	String
Default	null

Deploy EJB option - 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.

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, they are deprecated.

Deploy EJB option - Database type is the same as the `ejbdeploy` command parameter `-dbvendor` *`name`*.

Data type	String
Default	DB2UDB_V82

Deploy EJB option - 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"
```

Deploy EJB option - Database schema is the same as the `ejbdeploy` command parameter `-dbschema` *`"name"`*.

Data type	String
Default	null

Bind listeners for message-driven beans settings

Use this panel to specify bindings for message-driven beans in your application or module.

To view this administrative console panel, click **Applications > Enterprise Applications >** *application_name* **> Message Driven Bean listener bindings**. This panel is the same as the **Bind listeners for message-driven beans** panel 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 any of the following Java Message Service (JMS) providers:

- Version 5 default messaging
- WebSphere MQ
- Generic

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** panel.
- If the module containing the message-driven bean is deployed on a 5.x deployment target and a listener port is not specified, then a validation error is displayed after you click **Next**.
- 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. Specify a listener port name or select a target resource JNDI name for an activation specification.
4. If you are defining a binding for an activation specification, optionally specify the following:
   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.
5. Click **Apply**.
6. Click **OK**.

EJB module:

Specifies the name of the module that contains the enterprise bean.

EJB:

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.

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`.

Example: Installing an EAR file using the default bindings

If application bindings were not specified for all enterprise beans or resources in an 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.

An example of a simple `.ear` file installation using the default bindings follows:

1. Go to the Preparing for application install pages.

   Click **Applications > Install New Application** in the console navigation tree.
2. For **Path to the new application**, specify the full path name of the `.ear` file.

   For this example, the base file name is `my_appl.ear` and the file resides on a server at `C:\sample_apps`.
3. For **How do you want to install the application**, select **Show me all installation options and parameters**.
4. Click **Next**.
5. On the second Preparing for application installation page, select **Generate default bindings** and click **Next**.

   Using the default bindings causes any incomplete bindings in the application to be filled in with default values. Existing bindings are not changed. By choosing this option, you can skip many of the steps on the Install New Application page and go directly to the Summary step.
6. If application security warnings are displayed, read the warnings and click **Continue**.
7. On the Install New Application page, click step 2, **Manage modules**, and verify the cell, node, and server onto which the application files will install.
   a. On the **Manage modules** panel, select the server onto which the application files will install from the **Clusters and Servers** list, click **Module** to select all of the application modules, and click **Next**.

      On the **Manage modules** panel, 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.
8. On the Install New Application page, click the step number beside **Summary**, the last step.
9. On the Summary panel, click **Finish**.

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 > 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.

Installing J2EE modules with JSR-88

You can install Java 2 Platform, Enterprise Edition (J2EE) modules on an application server provided by a WebSphere Application Server product using the J2EE Deployment API Specification (JSR-88).

JSR-88 defines standard application programming interfaces (APIs) to enable deployment of J2EE applications and stand-alone modules to J2EE product platforms. The J2EE Deployment Specification Version 1.1 is available at http://java.sun.com/j2ee/tools/deployment/reference/docs/index.html as part of the 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/.

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 J2EE product platform. The tool provider typically supplies software tools and an integrated development environment (IDE) for developing and assembly of J2EE application modules. The J2EE platform provides application management functions that deploy, undeploy, start, stop, and otherwise manage J2EE applications.

WebSphere Application Server is a J2EE 1.4 specification-compliant platform that implements the JSR-88 APIs. Complete the following steps to deploy (install) J2EE modules on an application server provided by the WebSphere Application Server platform.

1. Code a Java program that can access the JSR-88 DeploymentManager class for WebSphere Application Server.
   a. Write code that finds the JAR manifest file key J2EE-DeploymentFactory-Implementation-Class. Under JSR-88, your code finds the DeploymentFactory using the JAR manifest file key J2EE-DeploymentFactory-Implementation-Class. For WebSphere Application Server, the application management JAR file containing this key and providing support is *app_server_root*/`lib/wjmxapp.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. For example:

```
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;

// Get the DeploymentFactory implementation class from the MANIFEST.MF file.
JarFile wjmxappJar = new JarFile(new File(wasHome + "/lib/wjmxapp.jar"));
java.util.jar.Manifest manifestFile = wjmxappJar.getManifest();
Attributes attributes = manifestFile.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 URL, 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 WebSphere Application Server. The WebSphere Application Server URL for deployment has the 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.

The URL for deployment can have an optional parameter *connectorType*. For example, to use the RMI connector to access *myserver*, code the URL as follows:

```
"deployer:WebSphere:myserver:2809?connectorType=RMI"
```

2. **Optional:** Code a Java program that can customize or deploy J2EE applications or modules using the JSR-88 support provided by WebSphere Application Server.
3. Start the deployed J2EE applications or standalone J2EE modules using the JSR-88 API used to start applications or modules.

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 J2EE applications or standalone modules during deployment using the DConfigBean class in the Java 2 Platform, Enterprise Edition (J2EE) Deployment API Specification (JSR-88).

This topic assumes that you are deploying (installing) J2EE modules on an application server provided by the WebSphere Application Server platform 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/.

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.Hashtable objects. The hashtable entries contain configuration attributes, for which your code can get and set values.

1. Write code that installs J2EE modules on an application server using JSR-88.
2. Write code that accesses DConfigBeans generated by WebSphere Application Server during JSR-88 deployment. You (or a deployer) can then customize the accessed DConfigBeans instances. The following pseudocode shows how a J2EE tool provider can get DConfigBean instance attributes generated by WebSphere Application Server 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 table.
   int index = 0;
   Hashtable tbl = (Hashtable)
      ipDesc.getIndexedReadMethod().invoke
         (dcBean, new Object[]{new Integer(index)});

   while (tbl != null)
   {
      // Iterate over the hashtable and set values for attributes.

      // Set the table back into the DCBean.
      ipDesc.getIndexedWriteMethod().invoke
         (dcBean, new Object[]{new Integer(index), tbl});

      // Get the next entry in the indexed property
      tbl = (Hashtable)
         ipDesc.getIndexedReadMethod().invoke
            (dcBean, new Object[]{new Integer(++index)});
   }
}
```

Enterprise application collection

Use this page to view and manage enterprise applications.

This page lists installed enterprise applications. System applications, which are central to the product, are not shown in the list because users cannot edit them. Examples of system applications include *isclite*, *managementEJB* and *filetransfer*.

To view this administrative console page, click **Applications > Enterprise Applications**.

To view the values specified for an application's 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 an installed enterprise application, enable the **Select** check box beside the application name in the list and click a button:

Button	Resulting action
Start	Attempts to run the application. After the application starts up successfully, the state of the application changes to *Started* if the application starts up on all deployment targets, else the state changes to *Partial Start*.
Stop	Attempts to stop the processing of the application. After the application stops successfully, the state of the application changes to *Stopped* if the application stops on all deployment targets, else the state changes to *Partial Stop*.
Install	Opens a wizard that helps you deploy an application or a module such as a `.jar`, `.war` or `.rar` file onto a server or a cluster.
Uninstall	Deletes the application from the WebSphere Application Server configuration repository and deletes the application binaries from the file system of all nodes where the application modules are installed after the configuration is saved and synchronized with the nodes.

Button	Resulting action
Update	Opens a wizard that helps you update application files deployed on a server. You can update the full application, a single module, a single file, or part of the application. If a new file or module has the same name as a file or module already existing on the server, the new file or module replaces the existing file or module. If the new file or module does not exist on the server, it is added to the deployed application.
Remove File	Deletes a file of the deployed application or module. **Remove File** deletes a file from the configuration repository and from the file system of all nodes where the file is installed.
Export	Accesses the Export Application EAR files page, which you use to export an enterprise application to an EAR file at a location of your choice. Use the **Export** action to back up a deployed application and to preserve its binding information.
Export DDL	Accesses the Export Application DDL files page, which you use to export DDL files (`Table.ddl`) in the EJB modules of an enterprise application to a location of your choice.

These buttons are not available when this page is accessed from an application server settings page. When this page is accessed from an application server settings page, it is entitled the Installed applications page.

Name

Specifies the name of the installed (or deployed) application. Application names must be unique within a cell and cannot contain an unallowed character.

Application Status

Indicates whether the application deployed on the application server is started, stopped, or unavailable.

	**Started**	Application is running.
	**Partial Start**	Application is in the process of changing from a *Stopped* state to a *Started* 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 *Started* state to a *Stopped* state. Application has not stopped running yet.
	**Unavailable**	Status cannot be determined. An application with an unavailable status might, in fact, be running but have an unavailable status because the server running the administrative console cannot communicate with the server running the application.
	**Not applicable**	Application does not provide information as to whether it is running.

Startup order

Specifies the order in which applications are started when the server starts. The application with the lowest startup order is started first.

This table column is available only when this page is accessed from an application server settings page; thus when this page is entitled the Installed applications page.

Enterprise application settings

Use this page to configure an enterprise application.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name*.

Name
Specifies a logical name for the application. An application name must be unique within a cell and cannot contain an unallowed character.

An application name cannot begin with a period (**.**), cannot contain leading or trailing spaces, and cannot contain any of the following characters:

Unallowed 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	

Data type String

Application reference validation
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.

The resource can be defined on the server, its node, cell or the cluster if the server belongs to a cluster. Select **Don't validate** for no resource validation, **Issue warnings** for warning messages about incorrect resource references, or **Stop installation if validation fails** to stop operations that fail as a result of incorrect resource references.

This **Application reference validation** setting is the same as the **Validate input off/warn/fail** field on the application installation and update wizards.

Data type String
Default Issue warnings

Configuring an application

You can change the configuration of an application or module deployed on a server.

You can change the contents of and deployment descriptors for an application or module before deployment, such as in an assembly tool. However, it is assumed that the module is already deployed on a server.

Changing an application or module configuration consists of one or more of the following:
- Changing the settings of the application or module.
- Removing a file from an application or module.

- Updating the application or its modules.

This topic describes how to change the settings of an application or module using the administrative console.

- View current settings of the application or module.

  Click **Applications > Enterprise Applications >***application_name* to access the settings page for the enterprise application.

  Many application or module settings are available on other console pages that you can access by clicking links on the settings page for the enterprise application. For detailed information on the settings and allowed values, examine the online help for the console pages. When you installed the application or module, you specified most of the settings values.
- Map each module of your application to a target server.

  Specify the application servers or Web servers onto which to install modules of your application.
- Change how quickly your application starts compared to other applications or to the server.
- Configure the use of binary files.
- Change how your application or Web modules use class loaders.
- Map a virtual host for each Web module of your application. Configuring virtual hosts provides information on virtual hosts.
- Change application bindings or other settings of the application or module.
  1. Click **Applications > Enterprise Applications >** *application_name* **>** *property_or_item_name* in the console navigation tree. From the application settings page, you can access console pages for further configuring of the application or module.
     - Target specific application status
     - Security role to user/group mapping
     - View deployment descriptor
     - Application scope resources
     - Resource references
     - EJB references
     - Shared library references
     - Initial parameters for servlets
     - Session management
     - Context root for Web modules
     - JSP reloading options for Web modules
     - Environment entries for Web modules
     - Virtual hosts
     - Application profiles
     - 2.x CMP bean data sources
     - 2.x entity bean data sources.
     - EJB JNDI names
     - Correct use of system identity
     - Provide JMS and EJB endpoint URL information
     - Publish WSDL files
     - Provide HTTP endpoint URL information
     - Web modules
     - EJB modules
  2. Change the values for settings as needed, and click **OK**.
- **Optional:** Configure the application so it does not start automatically when the server starts. By default, an installed application starts when the server on which the application resides starts. You can configure the target mapping for the application so the application does not start automatically when the server starts. To start the application, you must then start it manually.

- If the installed application or module uses a resource adapter archive (RAR file), ensure that the **Classpath** setting for the RAR file enables the RAR file to find the classes and resources that it needs. Examine the **Classpath** setting on the console Resource adapter settings page.

The application or module configuration is changed. The application or standalone Web module is restarted so the changes take effect.

Save changes to your administrative configuration.

Application bindings

Before an application that is installed on an application server can start, all enterprise bean (EJB) references and resource references defined in the application must be bound to the actual artifacts (enterprise beans or resources) defined in the application server.

When defining bindings, you specify Java Naming and Directory Interface (JNDI) names for the referenceable and referenced artifacts in an application. The jndiName values specified for artifacts must be qualified lookup names. An example referenceable artifact is an EJB defined in an application. An example referenced artifact is an EJB or a resource reference used by the application. Binding definitions are stored in the `ibm-`*`xxx`*`-bnd.xmi` files of an application. The *xxx* can be `ejb-jar`, `web`, `application` or `application-client`.

This topic provides the following information about bindings:

Times when bindings can be defined

You can define bindings at the following times:

- During application development

  An application developer can create binding definitions in `ibm-`*`xxx`*`-bnd.xmi` files using a tool such as an IBM Rational developer tool. The developer then gives an enterprise application (`.ear` file) complete with bindings to an application assembler or deployer. When assembling the application, the assembler does not modify the bindings. Similarly, when installing the application onto a server supported by WebSphere Application Server, the deployer does not modify or override the bindings or generate default bindings unless changes to the bindings are necessary for successful deployment of the application.
- During application assembly

  An application assembler can define bindings when modifying deployment descriptors of an application. Bindings are specified in the **WebSphere Bindings** section of a deployment descriptor editor. Modifying the deployment descriptors might change the binding definitions in the `ibm-`*`xxx`*`-bnd.xmi` files created when developing an application. After defining the bindings, the assembler gives the application to a deployer. When installing the application onto a server supported by WebSphere Application Server, the deployer does not modify or override the bindings or generate default bindings unless changes to the bindings are necessary for successful deployment of the application.
- During application installation

  An application deployer or server administrator can modify the bindings when installing the application onto a server supported by WebSphere Application Server using the administrative console. New binding definitions can be specified on the install wizard pages.

  If the deployer or administrator selects to override any existing bindings or to generate default bindings during application installation, default bindings are assigned to the application and new bindings might need to be specified using the console.

Selecting **Generate Default Bindings** during application installation causes any incomplete bindings in the application to be filled in with default values. Existing bindings are not changed.

Restriction: Bindings can be defined or overridden during application installation for all modules except application clients. For clients, you must define bindings for application client modules during assembly and store the bindings in the `ibm-application-client-bnd.xmi` file.

- During configuration of an installed application

  After an application is installed onto a server supported by WebSphere Application Server, an application deployer or server administrator can modify the bindings by changing values in administrative console pages such as those accessed from the settings page for the enterprise application.

Required bindings

Before an application can be successfully deployed, bindings must be defined for references to the following artifacts:

EJB JNDI names

For each enterprise bean (EJB), you must specify a JNDI name. The name is used to bind an entry in the global JNDI name space for the EJB home object. An example JNDI name for a *Product* EJB in a *Store* application might be `store/ejb/Product`. The binding definition is stored in the `META-INF/ibm-ejb-jar-bnd.xmi` file.

If a deployer chooses to generate default bindings when installing the application, the install wizard assigns EJB JNDI names having the form *prefix*/*EJB_name* to incomplete bindings. The default prefix is `ejb`, but can be overridden. The *EJB_name* is as specified in the deployment descriptor `<ejb-name>` tag.

During and after application installation, EJB JNDI names can be specified on the Provide JNDI names for beans panel. After installation, click **Applications > Enterprise Applications >** *application_name* **> EJB JNDI names** in the administrative console.

Data sources for entity beans

Entity beans such as container-managed persistence (CMP) beans store persistent data in data stores. With CMP beans, an EJB container manages the persistent state of the beans. You specify which data store a bean uses by binding an EJB module or an individual EJB to a data source. Binding an EJB module to a data source causes all entity beans in that module to use the same data source for persistence.

An example JNDI name for a *Store* data source in a *Store* application might be `store/jdbc/store`. The binding definition is stored in IBM binding files such as `ibm-ejb-jar-bnd.xmi`. A deployer can also specify whether authentication is handled at the container or application level.

If a deployer chooses to generate default bindings when installing the application, the install wizard generates the following for incomplete bindings:

- For EJB 2.x `.jar` files, connection factory bindings based on the JNDI name and authorization information specified
- For EJB 1.1 `.jar` files, data source bindings based on the JNDI name, data source user name and password specified

The generated bindings provide default connection factory settings for each EJB 2.x `.jar` file and default data source settings for each EJB 1.1 `.jar` file in the application being installed. No bean-level connection factory bindings or data source bindings are generated unless they are specified in the custom strategy rule supplied during default binding generation.

During and after application installation, data sources can be mapped to 2.x entity beans on the 2.x CMP bean data sources panel and on the 2.x entity bean data sources panel. After installation, click **Applications > Enterprise Applications >** *application_name* in the administrative console, then select **2.x CMP bean data sources** or **2.x entity bean data sources**. Data sources can be

mapped to 1.x entity beans on the Map data sources for all 1.x CMP beans panel and on the Provide default data source mapping for modules containing 1.x entity beans panel. After installation, access console pages similar to those for 2.x CMP beans, except click links for 1.x CMP beans.

Backend ID for EJB modules

If an EJB `.jar` file that defines CMP beans contains mappings for multiple backend databases, specify the appropriate backend ID that determines which persister classes are loaded at run time.

Specify the backend ID during application installation. You cannot select a backend ID after the application is installed onto a server.

To enable backend IDs for individual EJB modules:

1. During application installation, select **Deploy enterprise beans** on the Select installation options panel. Selecting **Deploy enterprise beans** enables you to access the **Provide options to perform the EJB Deploy** panel.
2. On the **Provide options to perform the EJB Deploy** panel, set the database type to "" (null).

During application installation, if you select **Deploy enterprise beans** on the Select installation options panel and specify a database type for the EJB deployment tool on the **Provide options to perform the EJB Deploy** panel, previously defined backend IDs for all of the EJB modules are overwritten by the chosen database type.

The default database type is `DB2UDB_V81`.

For information on backend databases, refer to EJB deployment tool. For information on EJB Deploy options, refer to The ejbdeploy command.

EJB references

An enterprise bean (EJB) reference is a logical name used to locate the home interface of an enterprise bean. EJB references are specified during deployment. At run time, EJB references are bound to the physical location (global JNDI name) of the enterprise beans in the target operational environment. EJB references are made available in the `java:comp/env/ejb` Java naming subcontext.

For each EJB reference, you must specify a JNDI name. An example JNDI name for a *Supplier* EJB reference in a *Store* application might be `store/ejb/Supplier`. The binding definition is stored in IBM binding files such as `ibm-ejb-jar-bnd.xmi`. When the referenced EJB is also deployed in the same application server, you can specify a server-scoped JNDI name. But if the referenced EJB is deployed on a different application server or if `ejb-ref` is defined in an application client module, then you should specify the global cell-scoped JNDI name.

If a deployer chooses to generate default bindings when installing the application, the install wizard binds EJB references as follows: If an `<ejb-link>` is found, it is honored. If the `ejb-name` of an EJB defined in the application matches the `ejb-ref` name, then that EJB is chosen. Otherwise, if a unique EJB is found with a matching home (or local home) interface as the referenced bean, the reference is resolved automatically.

During and after application installation, EJB reference JNDI names can be specified on the Map EJB references to beans panel. After installation, click **Applications > Enterprise Applications >** *application_name* **> EJB references** in the administrative console.

For more information, refer to EJB references .

Resource references

A resource reference is a logical name used to locate an external resource for an application. Resource references are specified during deployment. At run time, the references are bound to the physical location (global JNDI name) of the resource in the target operational environment. Resource references are made available as follows:

Resource reference type	Subcontext declared in

Java DataBase Connectivity (JDBC) data source	java:comp/env/jdbc
JMS connection factory	java:comp/env/jms
JavaMail connection factory	java:comp/env/mail
Uniform Resource Locator (URL) connection factory	java:comp/env/url

For each resource reference, you must specify a JNDI name. If a deployer chooses to generate default bindings when installing the application, the install wizard generates resource reference bindings derived from the <res-ref-name> tag, assuming that the java:comp/env name is the same as the resource global JNDI name.

During application installation, resource reference JNDI names can be specified on the Map resource references to references panel. 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. Each resource reference defined in an application must be bound to a resource defined in your WebSphere Application Server configuration. After installation, click **Applications > Enterprise Applications >** *application_name* **> Resource references** in the administrative console to access the Resource references panel.

Virtual host bindings for Web modules

You must bind each Web module to a specific virtual host. The binding informs a Web server plug-in that all requests that match the virtual host must be handled by the Web application. An example virtual host to be bound to a *Store* Web application might be store_host. The binding definition is stored in IBM binding files such as WEB-INF/ibm-web-bnd.xmi.

If a deployer chooses to generate default bindings when installing the application, the install wizard sets the virtual host to default_host for each .war file.

During and after application installation, you can map a virtual host to a Web module defined in your application. On the Map virtual hosts for Web modules panel, specify a virtual host. 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. For example, an external URL for a Web artifact such as a JSP file is http://*host_name*:*virtual_host_port*/*context_root*/*jsp_path*. After installation, click **Applications > Enterprise Applications >** *application_name* **> Virtual hosts** in the administrative console.

Message-driven beans

For each message-driven bean, you must specify a queue or topic to which the bean will listen. A message-driven bean is invoked by a Java Messaging Service (JMS) listener when a message arrives on the input queue that the listener is monitoring. A deployer specifies a listener port or JNDI name of an activation specification as defined in a connector module (.rar file) under **WebSphere Bindings** on the **Beans** page of an assembly tool EJB deployment descriptor editor. 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.

If a deployer chooses to generate default bindings when installing the application, the install wizard assigns JNDI names to incomplete bindings.

- For EJB 2.x message-driven beans deployed as JCA 1.5-compliant resources, the install wizard assigns JNDI names corresponding to activationSpec instances in the form eis/MDB_ejb-name.
- For EJB 2.x 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.

During application installation using the administrative console, you can specify a listener port name or an activation specification JNDI name for every message-driven bean on the panel **Bind listeners for message-driven beans**. A listener port name must be provided when using the JMS providers: Version 5 default messaging, WebSphere MQ, or generic. An activation specification

must be provided when the application's resources are configured using the default messaging provider or any generic J2C resource adapter that supports inbound messaging. If neither is specified, then a validation error is displayed after you click **Finish** on the Summary panel. Also, if the module containing the message-driven bean is deployed on a 5.x deployment target and a listener port is not specified, then a validation error is displayed after you click **Next**.

After application installation, you can specify JNDI names and configure message-driven beans on console pages under **Resources > JMS Providers** or under **Resources > Resource Adapters**. For more information, refer to "Using asynchronous messaging" on page 811.

Message destination references

A message destination reference is a logical name used to locate an enterprise bean in an EJB module that acts as a message destination. Message destination references exist only in J2EE 1.4 artifacts such as--

- J2EE 1.4 application clients
- EJB 2.1 projects
- 2.4 Web applications

If multiple message destination references are associated with a single message destination link, then a single JNDI name for an enterprise bean that maps to the message destination link, and in turn to all of the linked message destination references, is collected during deployment. At run time, the message destination references are bound to the administered message destinations in the target operational environment.

If a message destination reference and a message-driven bean are linked by the same message destination, both the reference and the bean should have the same destination JNDI name. When both have the same name, only the destination JNDI name for the message-driven bean is collected and applied to the corresponding message destination reference.

If a deployer chooses to generate default bindings when installing the application, the install wizard assigns JNDI names to incomplete message destination references as follows: If a message destination reference has a `<message-destination-link>`, then the JNDI name is set to `ejs/message-destination-linkName`. Otherwise, the JNDI name is set to `eis/message-destination-refName`.

Other bindings that might be needed

Depending on the references in and artifacts used by your application, you might need to define bindings for references and artifacts not listed in this article.

Configuring application startup

You can configure the startup behavior of an application. The values set affect how quickly an application starts and what occurs when an application starts.

This topic assumes that your application or module is already deployed on a server.

This topic also assumes that your application or module is configured to start automatically when the server starts. By default, an installed application starts when the server on which the application resides starts.

This topic describes how to change the settings of an application or module using the administrative console.

1. Click **Applications > Enterprise Applications >** *application_name* **> Startup behavior** in the console navigation tree.
2. Specify the startup order for the application.

If your application starts automatically when its server starts, the value for **Startup order** controls how quickly the application starts. **Startup order** specifies the order in which applications are started when the server starts. The application with the lowest startup order, or starting weight, is started first.

3. Specify whether the application must initialize fully before its server is considered started.

   If your application starts automatically when its server starts, **Launch application before server completes startup** specifies whether the application must initialize fully before its server is considered started. Background applications can be initialized on an independent thread, thus allowing the server startup to complete without waiting for the application. This setting applies only if the application is run on a Version 6 (or later) application server.

4. Specify whether to create MBeans for resources such as servlets or JavaServer Pages (JSP) files within an application when the application starts.

   The default for **Create MBeans for resources** is to create MBeans.

The application or module configuration is changed. The application or standalone Web module is restarted so the changes take effect.

Save changes to your administrative configuration.

Startup behavior settings

Use this page to configure when an application starts compared to other applications and to the server, and to configure whether MBeans for resources are created when an application starts.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Startup behavior**.

Startup order:

Specifies the order in which applications are started when the server starts. The startup order is like a starting weight. The application with the lowest starting weight is started first.

Data type	Integer
Default	1
Range	0 to 2147483647

Launch application before server completes startup:

Specifies whether the application must initialize fully before the server starts.

The default setting of `false` indicates that server startup will not complete until the application starts.

A setting of `true` informs the product that the application might start on a background thread and thus server startup might continue without waiting for the application to start. Thus, the application might not be ready for use when the application server starts.

This setting applies only if the application is run on a Version 6 application server.

Data type	Boolean
Default	false

Create MBeans for resources:

Specifies whether to create MBeans for various resources (such as servlets or JSP files) within an application when the application starts. The default is to create MBeans.

Data type	Boolean
Default	true

Configuring binary location and use

You can designate where binary files (binaries) used by your application reside, whether the product distributes binaries for you automatically, and otherwise configure the use of binaries.

This topic assumes that your application or module is already deployed on a server.

This topic describes how to change the settings of an application or module using the administrative console.

1. Click **Applications > Enterprise Applications >** *application_name* **> Application binaries** in the console navigation tree. The Application binaries page is displayed.
2. Specify the directory to hold the application binaries.

   The default is `APP_INSTALL_ROOT/`*`cell_name`*, where the `APP_INSTALL_ROOT` variable is *app_server_root*`/installedApps`. For example:

   `C:\WebSphere\AppServer\profiles\`*`profile_name`*`\installedApps\`*`cell_name`*

   Refer to "Application binary settings" on page 65 for a detailed description of the **Location (full path)** setting.
3. Specify the bindings, extensions, and deployment descriptors that an application server uses.

   By default, an application server uses the bindings, extensions, and deployment descriptors located with the application deployment document, the `deployment.xml` file.

   To specify that the application server use the bindings, extensions, and deployment descriptors located in the application archive (EAR) file, select **Use configuration information in binary**. Select this setting for applications installed on 6.x deployment targets only. This setting is not valid for applications installed on 5.x deployment targets.
4. Specify whether the product distributes application binaries automatically to other nodes on the cell.

   By default, **Enable binary distribution, expansion and cleanup post uninstallation** is selected and binaries are distributed automatically.

   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.

   Important: 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.
5. Specify access permissions for binaries.
   a. Ensure that the **Enable binary distribution, expansion and cleanup post uninstallation** option is enabled. That option must be enabled to specify access permissions for binaries.
   b. For **File permissions**, specify a string that defines access permissions for binaries that are expanded in the named location.

      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 drop-down list. Drop-down list selections overwrite file permissions set in the text field.

      For details on **File permissions**, refer to "Application binary settings" on page 65.
6. Click **OK**.

The application or module configuration is changed. The application or standalone Web module is restarted so the changes take effect.

Save changes to your administrative configuration.

Application binary settings

Use this page to configure the location and distribution of application binary files.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Application binaries**.

Location (full path):

Specifies the directory to which the application EAR file is installed. This **Location** setting is the same as the **Directory to install application** field on the application installation and update wizards.

The default value is the value of `APP_INSTALL_ROOT/`*`cell_name`*, where the `APP_INSTALL_ROOT` variable is *`app_server_root`*`/installedApps`; for example, *`app_server_root`*`/installedApps/`*`cell_name`*.

You can specify an absolute path or use a pathmap variable such as `${MY_APPS}`. You can use a pathmap variable in any installation.

Data type	String
Units	Full path name

Use configuration information in binary:

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 application resource (EAR) file.

This **Use configuration information in binary** setting is the same as the **Use binary configuration** field on the application installation and update wizards. Select this setting for applications installed on 6.x deployment targets only. This setting is not valid for applications installed on 5.x deployment targets.

Data type	Boolean
Default	false

Enable binary distribution, expansion and cleanup post uninstallation:

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. The binaries are deleted when you uninstall and save changes to the configuration, and, on the Network Deployment product, 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.

Important: 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 **Enable binary distribution, expansion and cleanup post uninstallation** setting is the same as the **Distribute application** field on the application installation and update wizards.

Data type	Boolean
Default	true

File permissions:

Specifies access permissions for application binaries for installed applications that are expanded to the directory specified.

The **Enable binary distribution, expansion and cleanup post uninstallation** 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 drop-down list. Drop-down list selections overwrite file permissions set in the text field.

You can set one or more of the following file permission strings in the drop-down list. Selecting multiple options combines the file permission strings.

Drop-down 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 drop-down 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.

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 following directory and file URIs are processed during a file permission operation:

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.

Windows You cannot unset read permission on a file on Windows platforms. With POSIX style permission bits, the bit for denoting `readable` on a file is 4, `writable` is 2, and `executable` is 1. Thus, permission of a file on a Windows platform is either 5 or 7. Also, in POSIX style there are `user`, `group` and `world` permissions. You can only set the `user` permission for a file on Windows platforms. The `group` and `world` permission bits are ignored.

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 permission** field on the application installation and update wizards.

Data type String

Application build level:

Specifies an uneditable string that identifies the build version of the application.

Data type String

Configuring the use of class loaders by an application

You can configure whether your application and Web modules use their own class loaders to load classes or use different class loaders, as well as configure the reloading of classes when application files are updated. Class loaders enable an application to access repositories of available classes and resources.

This topic assumes that your application or module is already deployed on a server.

Selection of class loaders to be used by an application and Web modules affects whether your application and its modules find the resources that they need to run effectively. You can select whether your application and Web modules use their own class loaders to load classes, or use a parent class loader.

Detailed information on class loaders is available in “Class loaders” on page 15, Chapter 5, “Class loading,” on page 15 and Troubleshooting class loaders.

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.

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.

You can also select whether classes are reloaded when application files are updated. For enterprise bean (EJB) modules or any non-Web modules, enabling class reloading causes the application server run time to stop and start the application to reload application classes. For Web modules such as servlets and JavaServer Pages (JSP) files, a Web container reloads a Web module only when the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file is set to `true`.

To configure use of class loaders by your application and Web modules, use the Class loading and update detection page of the administrative console.

1. Click **Applications > Enterprise Applications >** *application_name* **> Class loading and update detection** to access the settings page for an application class loader.
2. Specify whether to reload application classes when the application or its files are updated.

   By default, class reloading is not enabled. Select **Reload classes when application files are updated** 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.
3. 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.xmi`) 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).
4. 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 **Class 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 application class loader first	Causes the class loader to search in the application class loader first to load a class. By specifying `Classes loaded with application class loader first`, your application can override classes contained in the parent class loader. **Attention:** Specifying the `Classes loaded with application class loader first` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

5. 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**.

The application or module configuration is changed. The application or standalone Web module is restarted so the changes take effect.

Save changes to your administrative configuration.

Class loading and update detection settings

Use this page to configure use of class loaders by an application.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Class loading and update detection**.

Reload classes when application files are updated:

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

Select **Reload classes when application files are updated** to set reloadEnabled to `true` in the `deployment.xml` file for the application. If an application's class definition changes, the application server run time stops and starts the application to reload application classes.

For JavaServer Pages (JSP) files in a Web module, a Web container reloads JSP files only when the IBM extension jspReloadingEnabled in the jspAttributes of the `ibm-web-ext.xmi` file is set to `true`. You can enable JSP reloading during deployment on the JSP Reload Options panel.

Data type Boolean
Default false

Polling interval for updated files:

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.xmi`) file of the EAR file.

This **Polling interval for updated files** setting is the same as the **Reload interval in seconds** field on the application installation and update wizards.

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.

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

Data type	Integer
Units	Seconds
Default	3

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 application class loader first`, 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 application class loader first`. 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 WebSphere Application Server, set this application class loader mode to `Classes loaded with parent class loader first`. For your application to override the default configuration of Jakarta Commons Logging in WebSphere Application Server, your application must provide the configuration in a form supported by Jakarta Commons Logging and this class loader mode must be set to `Classes loaded with application class loader first`. Also, to override the default configuration, set the class loader mode for each Web module in your application so that the correct logger factory loads.

Data type	String
Default	Classes loaded with parent class loader first

WAR class loader policy:

Specifies whether to use a single class loader to load all WAR files of the application or to use a different class loader for each WAR file.

The options are `Class loader for each WAR file in application` and `Single class loader for application`. The default is to use a separate class loader to load each WAR file.

Data type	String
Default	Class loader for each WAR file in application

Manage modules settings

Use this panel to 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. A deployment target can be an application server, cluster of application servers or Web server.

To view this administrative console panel, click **Applications > Enterprise Applications >** *application_name* **> Manage modules**. This panel is the similar to the **Map modules to servers** panel on the application installation and update wizards.

On this panel, each **Module** must map to one or more desired targets, identified under **Server**. To change a mapping:

1. In the list of mappings, select the **Select** check box beside each module that you want mapped to the same target(s).
2. From the **Clusters and Servers** drop-down list, select one or more targets. Select only appropriate deployment targets for a module. Modules that use WebSphere Application Server Version 6.x features cannot be installed onto a Version 5.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 or cluster and the Web server together. The plug-in configuration file `plugin-cfg.xml` for that Web server will be generated based on the applications which are routed through it.
3. Click **Apply**.

If this Manage modules panel was accessed from a console enterprise application page for an already installed application, you can also use this panel 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, enable the **Select** check box beside the module name in the list and click a button:

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). 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 application is stopped, the module file is deleted from the node's file system, and the application is restarted.
Update	Opens a wizard that helps you update module 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. 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 application is stopped, the module file is updated on the node's file system, and the application is restarted. If the application is running on a node when the module file is added as a result of configuration synchronization then the newly added module is started without stopping and restarting the running application.
Remove File	Deletes a file from a module of a deployed application. 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 updated on the node as a result of configuration synchronization then the application is stopped, the module file is updated on the node's file system, and the application is restarted.

Clusters and Servers

Lists the names of available target servers and clusters. 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 can install an application, enterprise bean (EJB) module or Web module developed for a Version 5.x product on a 5.x or 6.x deployment target, provided the module--

- Does not support Java 2 Platform, Enterprise Edition (J2EE) 1.4;
- Does not call any 6.x runtime application programming interfaces (APIs); and
- Does not use any 6.x product features.

If the module supports J2EE 1.4, then you must install the module on a 6.x deployment target. If the module calls a 6.1.x API or uses a 6.1.x feature, then you must install the module on a 6.1.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 or 6.1.x 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 panel accessed from a console enterprise application page.

Server
Specifies the name of each server or cluster to which the module currently is mapped--that is, the deployment targets.

To change the deployment targets for a module, select one or more targets from the **Clusters and Servers** drop-down list and click **Apply**. The new mapping replaces the previous mapping.

Mapping modules to servers
Each module of a deployed application must be mapped to one or more target servers. The target server can be an application server or Web server.

You can map modules of an application or standalone Web module to one or more target servers during or after application installation using the console. This topic assumes that the module is already installed on a server and that you want to change the mappings.

Before you change a mapping, check the deployment targets. You must specify an appropriate deployment target for a module. Modules that use Version 6.x features cannot be installed onto a Version 5.x target server.

During application installation, different deployment targets might have been specified.

You use the Manage modules panel of the administrative console to view and change mappings. This panel is displayed during application installation using the console and, after the application is installed, can be accessed from the settings page for an enterprise application.

On the Manage modules panel, specify target servers where you want to install the modules contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that will serve as routers for requests to your application. The plug-in configuration file `plugin-cfg.xml` for each Web server is generated based on the applications which are routed through it.

1. Click **Applications > Enterprise Applications >** *application_name* **> Manage modules** in the console navigation tree. The Manage modules panel is displayed.
2. Examine the list of mappings. Ensure that each **Module** entry is mapped to the desired target(s), identified under **Server**.
3. Change a mapping as needed.
   a. Select each module that you want mapped to the same target(s). In the list of mappings, place a check mark in the **Select** check boxes beside the modules.
   b. From the **Clusters and Servers** drop-down list, select one or more targets. Use the **Ctrl** key to select multiple targets. For example, to have a Web server serve your application, use the **Ctrl** key

to select an application server 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.

c. Click **Apply**.

4. Repeat steps 2 and 3 until each module maps to the desired target(s).
5. Click **OK**.

The application or module configurations are changed. The application or standalone Web module is restarted so the changes take effect.

Save changes to your administrative configuration.

Mapping virtual hosts for Web modules

A virtual host must be mapped to each Web module of a deployed application. Web modules can be installed on the same virtual host or dispersed among several virtual hosts.

You can map a virtual host to a Web module during or after application installation using the console. This article assumes that the Web module is already installed on a server and that you want to change the mappings.

Before you change a mapping, check the virtual hosts definitions. You can install a Web module on any defined virtual host. To view information on previously defined virtual hosts, click **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`.

During application installation, a virtual host other than the one you want mapped to your Web module might have been specified.

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`, WebSphere Application Server 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.

Use the Virtual hosts page of the administrative console to view and change mappings. This page is displayed during application installation using the console and, after the application is installed, can be accessed from the settings page for an enterprise application.

On the Virtual hosts page, specify a virtual host for each Web module. Web modules of an application can be installed on the same virtual host or on different virtual hosts.

1. Click **Applications > Enterprise Applications >***application_name* **> Virtual hosts** in the console navigation tree. The Virtual hosts page is displayed.
2. Examine the list of mappings. Ensure that each **Web module** entry has the desired virtual host mapped to it, identified under **Virtual host**.

3. Change the mappings as needed.
   a. Select each Web module that you want mapped to a particular virtual host. In the list of mappings, place a check mark in the **Select** check boxes beside the Web modules.
   b. From the **Virtual host** drop-down list, select the desired virtual host. If you selected more than one virtual host in step 1:
      1) Expand **Apply Multiple Mappings**.
      2) Select the desired virtual host from the **Virtual host** drop-down list.
      3) Click **Apply**.
4. Repeat steps 2 and 3 until a desired virtual host is mapped to each Web module.
5. Click **OK**.

The application or Web module configurations are changed. The application or standalone Web module is restarted so the changes take effect.

After mapping virtual hosts, do the following:

1. Regenerate the plug-in configuration file.
   a. Click **Servers > Web servers**.
   b. Select the Web server for which you want to generate a plug-in.
   c. Click **Generate Plug-in**.
2. Save changes to your administrative configuration.

Virtual hosts settings

Use this panel 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 panel, click **Applications > Enterprise Applications >** *application_name* **> Virtual hosts**. This panel is the same as the **Map virtual hosts for Web modules** panel on the application installation and update wizards.

On this panel, 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**.

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.

Mapping properties for a custom login configuration

Use this page to view and manage the mapping properties for a custom login configuration.

To access the administrative console panel, complete the following steps:

1. Click **Applications > Enterprise applications >** *application_name*.
2. Under Enterprise Java Bean Properties, click **2.x entity bean data sources**.
3. For Container authorization, modify the authorization type by selecting you rEJB module and selecting **Container** from the Resource authorization menu.
4. Click **Apply**.
5. Under Specify authentication method, select **Use custom login configuration** and the name of the application login configuration.
6. Select the name of your EJB module.
7. Click **Apply**.
8. Click **Mapping properties** in the Resource authorization column. This property is not available until after you click apply in the previous step.

Name

Specifies the name for the mapping property.

Do not use the `MAPPING_ALIAS` property name because the name is reserved by the product.

Value

Specifies the value paired with the specified name.

Description

Provides additional information about the name and value pair.

Viewing deployment descriptors

A deployment descriptor is an extensible markup language (XML) file that specifies configuration and container options for an application or module.

This topic assumes that you have installed an application or module on a server and that you want to view its deployment descriptor.

When you create an application or module in an assembly tool such as the Application Server Toolkit (AST) or Rational Application Developer, the assembly tool creates deployment descriptor files for the application or module.

You can edit a deployment descriptor file manually. However, it is preferable to edit a deployment descriptor using an assembly tool deployment descriptor editor to ensure that the deployment descriptor has valid properties and that its references contain appropriate values.

After an application or module is installed on a server, you can view its deployment descriptor in the administrative console.

1. Access a deployment descriptor view.

   Click the navigational option stated in **Accessing a console view** to view the deployment descriptor for a given module:

Module	Deployment descriptor file	Accessing a console view
Enterprise application	`application.xml`	**Applications > Enterprise Applications >** *application_name* **> View deployment descriptor**
Web application	`WEB-INF/web.xml`	**Applications > Enterprise Applications >** *application_name* **> Manage Modules >** *module_name* **>** • **View deployment descriptor**
	`WEB-INF/portlet.xml`	**Applications > Enterprise Applications >** *application_name* **> Manage Modules >** *module_name* **>** • **View portlet deployment descriptor**
Enterprise bean	`ejb-jar.xml`	**Applications > Enterprise Applications >** *application_name* **> Manage Modules >** *module_name* **> View deployment descriptor**
Application client	`application-client.xml`	No console view
Web service	`webservices.xml`	**Applications > Enterprise Applications >** *application_name* **> Manage Modules >** *module_name* **>** • **View Web services client deployment descriptor extension** • **View Web services server deployment descriptor** • **View Web services server deployment descriptor extension** Viewing Web services deployment descriptors in the administrative console describes the views.
Resource adapter	`ra.xml`	**Resource Adapters > Resource Adapters >** *module_name* **> View deployment descriptor**

2. Click **Expand All** to view the deployment descriptor contents.

A deployment descriptor such as the following for the product DefaultApplication is displayed:

```
<application id="Application_ID" >
  <display-name> DefaultApplication.ear</display-name>
  <description> This is the IBM WebSphere Application Server Default Application.</<description>
  <module id="WebModule_1" >
    <web>
      <web-uri> DefaultWebApplication.war</web-uri>
      <context-root> /</context-root>
    </web>
  </module>
  <module id="EjbModule_1" >
    <ejb> Increment.jar</ejb>
```

```
  </module>
  <security-role id="SecurityRole_1130344639273" >
    <description> All Authenticated users role.</description>
    <role-name> All Role</role-name>
  </security-role>
</application>
```

Verify the deployment descriptor contents, including any configurations that it has for bindings, security roles, references to other resources, or Java Naming and Directory Interface (JNDI) names.

Change a deployment descriptor as needed in an assembly tool or using the console.

Starting or stopping applications

You can start an application that is not running (has a status of *Stopped*) or stop an application that is running (has a status of *Started*).

This topic assumes that the application is installed on a server. By default, the application starts automatically when the server starts.

You can start and stop applications manually using the following:
- Administrative console
- wsadmin startApplication and stopApplication commands
- Java programs that use ApplicationManager or AppManagement MBeans

This topic describes how to use the administrative console to start or stop an application.

1. Go to the Enterprise Applications page. Click **Applications > Enterprise Applications** in the console navigation tree.
2. Select the check box for the application you want started or stopped.
3. Click a button:

Option	Description
Start	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.
Stop	Stops the processing of the application and changes the state of the application to *Stopped*.

   To restart a running application, select the application you want to restart, click **Stop** and then click **Start**.

The status of the application changes and a message stating that the application started or stopped displays at the top the page.

You can configure an application so it does not start automatically when the server on which it resides starts. You then start the application manually using options described in this article.

If you want your application to start automatically when its server starts, you can adjust values that control how quickly the application or its server starts:

1. Go the settings page for your enterprise application. Click **Applications > Enterprise Applications >** *application_name* **> Startup behavior**.
2. Specify a different value for **Startup order**.

   This setting specifies the order in which applications are started when the server starts. The default value is 1 in a range from 0 to 2147483647. The application with the lowest starting weight is started first.

3. Specify a different value for **Launch application before server completes startup**.

   This setting specifies whether the application must initialize fully before its server starts. The default value of `false` prevents the server from starting completely until the application starts. To reduce the amount of time it takes to start the server, you can set the value to `true` and have the application start on a background thread, thus allowing server startup to continue without waiting for the application
4. Save the changes to the application configuration.

Disabling automatic starting of applications

You can enable and disable the automatic starting of an application. By default, an installed application starts automatically when the server on which the application resides starts.

This topic assumes that the application is installed on an application server and that the application starts automatically when the server starts.

This topic also assumes that you mapped the installed application to a server.

You might want an application to run only after you start it manually and not to run every time after the server starts. The target mapping for an application controls whether an application starts automatically when the server starts or requires you to start the application manually.

1. Go to the Target specific application status page for your application.

   Click **Applications > Enterprise Applications >** *application_name* **> Target specific application status**.
2. Select the target server on which the application resides.
3. Click **Disable Auto Start**.
4. Save changes to the administrative configuration.

The application does not start when its server starts. You must start the application manually.

To enable automatic starting of the application, do the following:
1. On the Target specific application status page for the application, select the target on which the application resides.
2. Click **Enable Auto Start**.
3. Save changes to the configuration.

Target specific application status

Use this page to view mappings of deployed applications or modules to servers or clusters.

Also use this page to enable or disable the automatic starting of an application when the server on which the application resides starts.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Target specific application status**.

Target

States the name of the target server or cluster to which the application or module maps. You specify the target on the Manage modules page accessed from the settings for an application.

Node

Specifies the node name if the target is a server.

Version

Specifies the version level of the target. The target can be a 5.x deployment target or a 6.x deployment target.

A *5.x deployment target* is a server or a cluster with at least one member on a WebSphere Application Server Version 5 product.

A *6.x deployment target* is a server or cluster with all members on a WebSphere Application Server Version 6 product.

An application, enterprise bean (EJB) module Session Initiation Protocol (SIP) module (SAR), or Web module developed for a Version 5.x product can reside on a 5.x or 6.x deployment target, provided the module--
- Does not support Java 2 Platform, Enterprise Edition (J2EE) 1.4;
- Does not call any 6.x runtime application programming interfaces (APIs); and
- Does not use any 6.x product features.

Similarly, a resource adapter (connector) module, or RAR file, developed for a Version 5.x product can reside on a 5.x or 6.x node, provided the module does not support Java Cryptography Architecture (JCA) 1.5 and does not call any 6.x runtime application programming interfaces (APIs). If the module supports JCA 1.5 or calls a 6.x API, then the module must reside on a 6.x node.

If JavaServer Pages (JSP) precompilation, EJB deployment (`ejbdeploy`), or Web Services deployment (`wsdeploy`) are enabled, then you can deploy applications to only those targets that have same product version as the deployment manager. If applications are targeted to servers that have an earlier version than the deployment manager, then you cannot deploy to those targets. Thus, if JSP precompilation, `ejbdeploy`, or `wsdeploy` are enabled, then you can deploy applications to only a 6.1 target.

Auto Start

Specifies whether the application modules installed on the target server are started (or enabled) when the server starts. This setting specifies the initial state of application modules. A `Yes` value indicates that the corresponding modules are enabled and thus are accessible when the server starts. A `No` value indicates that the corresponding modules are not enabled and thus are not accessible when the server starts.

By default, Auto Start is enabled. Thus, by default an installed application starts automatically when the server on which the application resides starts.

You can enable and disable the automatic starting of the application. To disable the automatic starting of the application, enable the **Select** check box beside the target server or cluster and click **Disable Auto Start**. When automatic starting is disabled, the application does not start when its server starts. To enable the automatic starting of the application, select the target and click **Enable Auto Start**.

Application Status

Indicates whether the application deployed on the application server is started, stopped, or unavailable.

	**Started**	Application is running.
	**Partial Start**	Application is in the process of changing from a *Stopped* state to a *Started* state. Application is starting to run but is not fully running yet. The application might be in the Partial Start state because one of its application servers is not started.
	**Stopped**	Application is not running.
	**Partial Stop**	Application is in the process of changing from a *Started* state to a *Stopped* state. Application has not stopped running yet.
	**Unavailable**	Status cannot be determined. An application with an unavailable status might, in fact, be running but have an unavailable status because the server running the administrative console cannot communicate with the server running the application.
	**Not applicable**	Application does not provide information as to whether it is running.

Exporting applications

You can export an enterprise application to a location of your choice.

Exporting applications enables you to back up your applications and preserve binding information for the applications. You might export your applications before updating installed applications or migrating to a later version of the WebSphere Application Server product.

1. Click **Applications > Enterprise Applications** in the console navigation tree to access the Enterprise Applications page.
2. Select the check box beside the application and click **Export**.
3. On the Export Application EAR Files page, click on the link to download the exported EAR file.
4. Use the browser dialogue to specify a location at which to save the exported EAR file.
5. Click **Back** to return to the Enterprise Applications page.

The file containing binding information is exported to the specified node and directory, and has the name *enterprise_application_name*.ear.

Exporting DDL files

You can export data definition language (DDL) files in the enterprise bean (EJB) modules of an application.

Exporting DDL (Table.ddl) files in the EJB modules of an application downloads the DDL files to a location of your choice.

1. Click **Applications > Enterprise Applications** in the administrative console navigation tree to access the Enterprise Applications page.
2. Place a check mark in the check box beside the application and click **Export DDL**. If the application has no DDL files in any of its EJB modules, then the message *No DDL files were found* is displayed at the top of the page. If the application has DDL files in its EJB modules, then a page listing DDL files in the format *application_name.ear*/*_module*.jar_Table.ddl is displayed.
3. Click on a file in the list and specify the location to which to download the file.

   Tip: Mozilla browsers might display the contents of the Table.ddl file instead of saving the file to disk. To save the file, edit the **Helper Application** preference settings of the Mozilla browser by adding a new type for DDL and specifying that you want to save DDL files to disk. That is, set MIME type = ddl and Extension = ddl.

The DDL file is downloaded to the specified location.

Updating applications

You can update application files deployed on a server.

Update your application or modules and reassemble them using an assembly tool. Typical tasks include adding or editing assembly properties, adding or importing modules into an application, and adding enterprise beans, Web components, and files.

Also, determine whether the updated files can be installed to your deployment targets. WebSphere Application Server Version 6.x supports Java 2 Platform, Enterprise Edition (J2EE) 1.4 enterprise applications and modules. If you are deploying J2EE 1.4 modules, ensure that the target server and its node support Version 6.x. The administrative console Server collection pages show the versions for

servers. You can deploy J2EE 1.4 modules to Version 6.x servers only. You cannot deploy J2EE 1.4 modules to servers on Version 5.x nodes. See “Installable module versions” on page 29 for details.

Updating consists of adding a new file or module to an installed application, or replacing or removing an installed application, file or module. After replacement of a full application, the old application is uninstalled. After replacement of a module, file or partial application, the old installed module, file or partial application is removed from the installed application.

1. Determine which method to use to update your application files. WebSphere Application Server provides several ways to update modules.
2. Update the application files using
   - Administrative console
   - wsadmin scripts
   - Java application programming interfaces
   - WebSphere rapid deployment of J2EE applications

   In some situations, you can update applications or modules without restarting the application server using hot deployment. Do not use hot deployment unless you are an experienced user and are updating applications in a development or test environment.
3. Start the deployed application files using
   - Administrative console
   - wsadmin startApplication
   - Java programs that use ApplicationManager or AppManagement MBeans

Save the changes to your administrative configuration.

Next, test the application. For example, point a Web browser at the URL for a deployed application (typically http://*hostname*:9060/*Web_module_name*, where *hostname* is your valid Web server and 9060 is the default port number) and examine the performance of the application. If the application does not perform as desired, edit the application configuration, then save and test it again.

Ways to update application files

You can update application files deployed on a server in several ways.

Table 3. Ways to update application files

Option	Method	Comments	Starting after update
Administrative console update wizard See “Updating applications with the console” on page 83.	Briefly, do the following: 1. Go to the Enterprise Applications page. Click **Applications > Enterprise Applications** in the console navigation tree. 2. Select the application to update and click **Update**. 3. On the Preparing for application update page, identify the application, module or files to update and click **Next**. 4. Complete steps in the update wizard and click **Finish**.	On the Preparing for application update page: • Use **Full application** to update an `.ear` file. • Use **Single module** to update a `.war`, `.sar`, enterprise bean `.jar`, or connector `.rar` file. • Use **Single file** to update a file other than an `.ear`, `.war`, `.sar`, EJB `.jar`, or `.rar` file. • Use **Partial application** to update or remove multiple files.	On the Enterprise Applications page, select the updated application and click **Start**.

Table 3. Ways to update application files (continued)

Option	Method	Comments	Starting after update
wsadmin scripts	Use the **update** command or the **updateInteractive** command in a script or at a command prompt. For more information on the **update** and **updateInteractive** commands, see the Commands for the AdminApp object topic.	Getting started with scripting provides an overview of wsadmin.	Start the application using the **invoke** command and the startApplication attribute. For more information about the **invoke** command, see the Commands for the AdminControl object topic.
Java application programming interfaces See Using administrative programs (JMX).	Update deployed applications by completing the steps in Managing applications through programming.	Update an application in the following ways: • Update the entire application • Add to, replace or delete multiple files in an application • Add a module to an application • Update a module in an application • Delete a module in an application • Add a file to an application • Update a file in an application • Delete a file in an application	• Invoke the AdminApp *startApplication* command. • Invoke the *startApplication* method on an ApplicationManager MBean using AdminControl.
WebSphere rapid deployment See topics under Rapid deployment of J2EE applications in this information center.	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.	WebSphere rapid deployment offers the following advantages: • 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.	Use any of the above options to start the application. Clicking **Start** on the Enterprise Applications page is the easiest option.
Hot deployment and dynamic reloading	Briefly, do the following: 1. Update your application (`.ear`), Web module (`.war`), enterprise bean `.jar` or HTTP plug-in configuration file. 2. Follow instructions in Hot deployment and dynamic reloading to update your file.	If you are new to WebSphere Application Server, use the administrative console to update applications. That option is easier. Hot deployment and dynamic reloading is more difficult to complete. You must directly manipulate the application or module file on the server where the application is deployed.	Use any of the above options to start the application. Clicking **Start** on the Enterprise Applications page is the easiest option.

You can update `.ear`, enterprise bean `.jar`, Web module `.war`, Session Initiation Protocol (SIP) module (`.sar`), connector `.rar`, application client `.jar`, and any other files used by an installed application.

If the application is updated while it is running, WebSphere Application Server automatically stops the application, updates the application logic and restarts the application. If the application does not start automatically, start it manually using one of the **Starting** options. 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*.

Updating applications with the console

Updating applications consists of adding a new file or module to an installed application, or replacing or removing an installed application, file or module.

Before you update the application files on a server, ensure that the files are assembled in deployable modules.

Next, refer to "Ways to update application files" on page 81 and decide how to update your application files. You can update enterprise applications or modules using the administrative console, the wsadmin tool, or Java MBean programming. These ways provide similar updating capabilities.

Further, ensure that the updated files can be installed to your deployment targets.

This topic describes how to update deployed applications or modules using the administrative console.

1. Back up the installed application.
   a. Go to the Enterprise Applications page of the administrative console. Click **Applications > Enterprise Applications** in the console navigation tree.
   b. Export the application to an EAR file. Select the application you want uninstalled and click **Export**. Exporting the application preserves the binding information.
2. With the application selected on the Enterprise Applications page, click **Update**. The Preparing for application update page is displayed.
3. Under **Specify the EAR, WAR, SAR or JAR module to upload and install**:
   a. Ensure that **Application to be updated** refers to the application to be updated.
   b. Under **Application update options**, select the installed application, module, or file that you want to update.

      The online help Preparing for application update settings provides detailed information on the options.
4. If you selected the **Replace the entire application** or **Replace or add a single module** option:
   a. Click **Next** to display a wizard for updating application files.
   b. Complete the steps in the update wizard.

      This update wizard, which is similar to the installation wizard, provides fields for specifying or editing application binding information. Refer to information on installing applications and on the settings page for application installation for guidance.

      Note that the installation steps have the merged binding information from the new version and the old version. If the new version has bindings for application artifacts such as EJB JNDI names, EJB references or resource references, then those bindings will be part of the merged binding information. If new bindings are not present, then bindings are taken from the installed (old) version. If bindings are not present in the old version and if the default binding generation option is enabled, then the default bindings will be part of the merged binding information.

      You can select whether to ignore bindings in the old version or ones in the new version.
5. Click **Finish**.
6. If you did not use the Manage modules page of the update wizard, after updating the application, map the installed application or module to servers.

   Use the Manage modules page accessed from the Enterprise Applications page.

a. Go to the Manage modules page. Click **Applications > Enterprise Applications >** *application_name* **> Manage modules**.
b. Specify the application server where you want to install modules contained in your application and click **OK**.

   You can deploy J2EE 1.4 modules to servers on Version 6.x nodes only.

After replacement of a full application, the old application is uninstalled. After replacement of a module, file or partial application, the old installed module, file or partial application is removed from the installed application.

After the application file or module installs successfully, do the following:

1. Save the changes to your configuration.

   When you update a full application in the single server (base) product, after you save the changes, the old version of the application is uninstalled and the new version is installed into the configuration. The application binaries for the old version are deleted from the destination directory and the new binaries are copied to the directory.
2. If needed, restart the application manually so the changes take effect.

   If the 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*.
3. If the application you are updating is deployed on a server that has its application class loader policy set to `Single`, restart the server.

Preparing for application update settings

Use this page to update enterprise applications, modules or files already installed on a server.

To view this administrative console page, do the following:
1. Click **Applications > Enterprise Applications**.
2. Select the installed application or module that you want to update.
3. Click **Update**.

Clicking **Update** displays a page that helps you update application files deployed in the cell. You can update the full application, a single module, a single file, or part of the application. If a new file or module has the same relative path as a file or module already existing on the server, the new file or module replaces the existing file or module. If the new file or module does not exist on the server, it is added to the deployed application.

Application to be updated

Specifies the name of the installed (or deployed) application that you selected on the Enterprise Applications page.

Replace the entire application

Under **Application update options**, specifies to replace the application already installed on the server with a new (updated) enterprise application `.ear` file.

After selecting this option, do the following:

1. Specify whether the `.ear` file is on a local or remote file system and the full path name of the application. The path provides the location of the updated `.ear` file before installation.

   Use **Local file system** if the browser and the updated files or modules are on the same machine, whether or not the server is on that machine too. **Local file system** is available for all update options.

   Use **Remote file system** if the application file resides on any node in the current cell context. Only `.ear`, `.jar`, `.sar`, or `.war` files are shown during the browsing.

Also use the **Remote file system** option to specify an application file already residing on the machine running the application server. For example, the field value might be *app_server_install_root*/installableApps/test.ear. If you are installing a standalone WAR module, then specify the context root as well.

Tip: During application installation, application files typically are uploaded from a client machine running the browser to the server machine running the administrative console, where they are deployed. In such cases, use the Web browser running the administrative console to select .ear, .war, .sar, or .jar modules to upload to the server machine. In some cases, however, the application files reside on the file system of any of the nodes in a cell. To have the application server install these files, use the **Remote file system** option.

2. If you are installing a standalone Web application (WAR) or a Session Initiation Protocol (SIP) module (SAR), specify the context root of the WAR or SAR file.

   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.
3. Specify whether to show only installation options that require you to supply information or to show all installation options.

   The **Prompt me only when additional information is required** option enables you to install your application more easily because you do not need to examine all available installation options.

   However, to use the **Generate default bindings** option, which might be the quickest and easiest option for installing your application, you must select the **Show me all installation options and parameters** and then select **Generate default bindings** on the next panel.
4. Click **Next** to display a wizard for updating application files. The update wizard, which is similar to the installation wizard, provides fields for specifying or editing application binding information. Complete the steps in the update wizard as needed.

When the full application is updated, the old application is uninstalled and the new application is installed. When the configuration changes are saved and subsequently synchronized, the application files are expanded on the node where application will run. If the application is running on the node while it is updated, then the application is stopped, application files are updated, and application is started.

Replace or add a single module

Under **Application update options**, specifies to replace a module in or add a module to an installed application. The module can be a Web module (.war file), enterprise bean module (EJB .jar file), SIP module (.sar file), or resource adapter module (connector .rar file).

After selecting this option, specify whether the module is on a local or remote file system and the full path name of the module. The path provides the location of the updated module before installation. For information on **Local file system** and **Remote file system**, refer to the description of **Replace the entire application** above.

To replace a module, the specified relative path (module URI) must match the path of the module to be updated in the installed application.

To add a new module to the installed application, the specified relative path must *not* match the path of a module in the installed application. The value specifies the desired path for the new module.

If you are installing a standalone Web or SIP module, specify a value for **Context root**. 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.

Next, specify whether to show only installation options that require you to supply information or to show all installation options.

After specifying the required information on the module, click **Next** to display a wizard for updating application files. The update wizard, which is similar to the installation wizard, provides fields for specifying or editing module binding information. Complete the steps in the update wizard as needed.

After a single module is added or updated, when configuration changes are saved, the new or updated module is stored in the deployed application in the WebSphere Application Server configuration repository. When these changes are synchronized with the node, the module is added or updated to the node's file system. If the application is running on the node when the module is added or updated, then one of the following occurs:

- For updates to a Web module, the running Web module is stopped, Web module files are updated, and then the Web module is started.
- For module additions, the added module is started on the application servers where the application is running after it is expanded on the node. An application restart is not necessary.
- If the class loader policy for the application is set to `Single` so that all modules share a class loader, then the entire application is stopped and restarted for module level changes.
- If the security provider configured with WebSphere Application Server does not support dynamic updates, then the entire application is stopped and restarted for module level changes.
- For all other updates to a module, the entire application is stopped, the module files are updated, then the entire application is started.

Replace or add a single file

Under **Application update options**, specifies to replace a file in or add a file to an installed application.

Use this option to update a file used by the application that is not an `.ear`, `.war`, `.sar`, `.rar` or, in some instances, a `.jar` file. You can use this option to add or update `.jar` files that are not defined as modules in the application. To update an `.ear`, file use the **Replace the entire application** option. To update a `.war` file, `.sar` file, `.rar` file, or `.jar` file that is defined as a module in the application, use the **Replace or add a single module** option.

After selecting this option, specify whether the file is on a local or remote file system and the full path name of the file. The path provides the location of the updated file before installation. For information on **Local file system** and **Remote file system**, refer to the description of **Replace the entire application** above.

For the relative path, specify a relative path to the file that starts from the root of the `.ear` 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`.

To replace a file, the relative path must match the path of the file to be updated in the installed application.

To add a new file to the installed application, the relative path must *not* match the path of a file in the installed application. The value specifies the desired path for the new file.

After you specify the file system and relative paths, click **Next**.

After a single file is added or updated, when configuration changes are saved, the new or updated file is stored in the deployed application in the WebSphere Application Server configuration repository. When these changes are synchronized with the node, the file is added or updated to the node's file system. If the application is running on the node when the file is added or updated, then one of the following occurs:

- When files are added at application metadata scope (`META-INF` directory) or updated at any application scope or in non-Web modules, the entire application is stopped, the file is added or updated, and then the entire application is restarted.
- When files are added at application non-metadata scope (outside of `META-INF` directory but not in any module), the changes are saved in the file system without restarting the running application.
- When files are added or updated to Web module metadata (`META-INF` or `WEB-INF` directory), the running Web module is stopped, the Web module file is added or updated, and then the Web module is started.

- For all other files in Web modules, the file is added or updated on the node's file system without stopping the application or any of its components.

Replace, add, or delete multiple files

Under **Application update options**, specifies to update multiple files of an installed application by uploading a compressed file. Depending on the contents of the compressed file, a single use of this option can replace files in, add new files to, and delete files from the installed application. Each entry in the compressed file is treated as a single file and the path of the file from the root of the compressed file is treated as the relative path of the file in the installed application.

After selecting this option, specify whether the compressed file is on a local or remote file system and the full path name of the compressed file. You will likely use **Local file system** because you are uploading a compressed file and remote browsing only works for `.ear`, `.sar`, `.war` or `.jar` files. Specify a valid compressed file format such as `.zip` or `.gzip`. The path provides the location of the compressed file before installation. This option unzips the compressed file into the installed application directory.

Use **Local file system** if the browser and the updated files or modules are on the same machine, whether or not the server is on that machine too. **Local file system** is available for all update options.

To replace a file, a file in the compressed file must have the same relative path as the file to be updated in the installed application.

To add a new file to the installed application, a file in the compressed file must have a different relative path than the files in the installed application.

The relative path of a file in the installed application is formed by concatenation of the relative path of the module (if the file is inside a module) and the relative path of the file from the root of the module separated by /.

To remove a file from the installed application, specify metadata in the compressed file using a file named `META-INF/ibm-partialapp-delete.props` at any archive scope. The `ibm-partialapp-delete.props` file must be an ASCII file that lists files to be deleted in that archive with one entry for each line. The entry can contain a string pattern such as a regular expression that identifies multiple files. The file paths for the files to be deleted must be relative to the archive path that has the `META-INF/ibm-partialapp-delete.props` file.

Level of files to delete	Metadata .props file to include in compressed file
Application	Include `META-INF/ibm-partialapp-delete.props` in the compressed file. In the metadata `.props` file, list files to be deleted. File paths are relative to the location of the `META-INF/ibm-partialapp-delete.props` file. For example, to delete a file named `utils/config.xmi` from the root of the `my.ear` file, include the line `utils/config.xmi` in the `META-INF/ibm-partialapp-delete.props` file.

Level of files to delete	Metadata .props file to include in compressed file
Module	Include *module_uri*/META-INF/ibm-partialapp-delete.props in the compressed file. To delete one file from a module, include the file path relative to the module in the metadata .props file. For example, to delete a/b/c.jsp from the my.jar module, include a/b/c.class in my.jar/META-INF/ibm-partialapp-delete.props file in the compressed file. To delete multiple files within a module, list the files to be deleted in the metadata .props file with one entry on each line. For example, to delete all JavaServer Pages (.jsp files) from the my.war file, include the line .*jsp in the my.war/META-INF/ibm-partialapp-delete.props file. The line uses a regular expression, .*jsp, to identify all .jsp files in my.war.

You can use a single partial application file to add, delete and update multiple files.

After you specify a file system path, click **Next**.

After a partial application update, when configuration changes are saved, the new or updated application file is stored in the deployed application in the WebSphere Application Server configuration repository. When these changes are synchronized with the node, the files are added or updated to the node's file system. Because the partial application option updates multiple files, the application components that are restarted are determined using individual files in the partial application.

An example of entries in a partial application compressed file follows:

```
util.jar
META-INF/ibm-partialapp-delete.props
foo.jar/com/mycomp/xyz.class
xyz.war/welcome.jsp
xyz.war/WEB-INF/web.xml
webmod.war/META-INF/ibm-partialapp-delete.props
```

For this example, the META-INF/ibm-partialapp-delete.props file contains the .*.dat and tools/test.jar files. The webmod.war/META-INF/ibm-partialapp-delete.props file contains the com/test/.*.jsp and WEB-INF/test.xmi files.

The partial application update option does the following:
- Adds or replaces util.jar in the deployed application.
- Adds or replaces com/mycomp/xyz.class inside the foo.jar file of the deployed application.
- Deletes *.dat files from the application, but not from any modules.
- Deletes tools/test.jar from the application.
- Adds or replaces welcome.jsp inside the xyz.war module of the deployed application.
- Replaces WEB-INF/web.xml inside the xyz.war module of the deployed application.
- Deletes com/test/*.jsp from the webmod.war module.
- Deletes WEB-INF/test.xmi from the webmod.war module.

Hot deployment and dynamic reloading

You can make various changes to applications and their modules without having to stop the server and start it again. Making these types of changes is known as *hot deployment and dynamic reloading*.

This topic assumes that your application files are deployed on a server and you want to upgrade the files.

See “Ways to update application files” on page 81 and determine whether hot deployment is the appropriate way for you to update your application files. Other ways are easier and hot deployment is appropriate only for experienced users.

Hot deployment is the process of adding new components (such as WAR files, EJB Jar files, enterprise Java beans, servlets, and JSP files) to a running server without having to stop the application server process and start it again.

Dynamic reloading is the ability to change an existing component without needing to restart the server in order for the change to take effect. Dynamic reloading involves:
- Changes to the implementation of a component of an application, such as changing the implementation of a servlet
- Changes to the settings of the application, such as changing the deployment descriptor for a Web module

As opposed to the changes made to a deployed application described in “Updating applications” on page 80, changes made using hot deployment or dynamic reloading do not use the administrative console or a wsadmin scripting command. You must directly manipulate the application files on the server where the application is deployed.

If the application you are updating is deployed on a server that has its application class loader policy set to `Single`, you might not be able to dynamically reload your application. At minimum, you must restart the server after updating your application.

1. Locate your expanded application files.

   The application files are in the directory you specified when installing the application or, if you did not specify a custom target directory, are in the default target directory, *app_server_root*/`installedApps`/*cell_name*. Your EAR file, `${APP_INSTALL_ROOT}`/*cell_name*/*application_name*`.ear`, points to the target directory. The `variables.xml` file for the node defines ${APP_INSTALL_ROOT}.

   It is important to locate the expanded application files because, as part of installing applications, a WebSphere application server unjars portions of the EAR file onto the file system of the computer that will run the application. These expanded files are what the server looks at when running your application. If you cannot locate the expanded application files, look at the binariesURL attribute in the `deployment.xml` file for your application. The attribute designates the location the run time uses to find the application files.

   For the remainder of this information on hot deployment and dynamic reloading, *application_root* represents the root directory of the expanded application files.

2. Locate application metadata files. The metadata files include the deployment descriptors (`web.xml`, `application.xml`, `ejb-jar.xml`, and the like), the bindings files (`ibm-web-bnd.xmi`, `ibm-app-bnd.xmi`, and the like), and the extensions files (`ibm-web-ext.xmi`, `ibm-app-ext.xmi`, and the like).

   Metadata XML files for an application can be loaded from one of two locations. The metadata files can be loaded from the same location as the application binary files (such as *application_root*`/META-INF`) or they can be loaded from the WebSphere configuration tree, `${CONFIG_ROOT}/cells/`*cell_name*`/applications /`*application_EAR_name*`/deployments/`*application_name*`/`. The value of the useMetadataFromBinary flag specified during application installation controls which location is used. If specified, the metadata files are loaded from the same location as the application binary files. If not specified, the metadata files are loaded from the application deployment folder in the configuration tree.

   For the remainder of this information, *metadata_root* represents the location of the metadata files for the specified application or module.

3. **Optional:** Examine the values specified for **Reload classes when application files are updated** and **Polling interval for updated files** on the settings page for your application’s class loader.

   If reloading of classes is enabled and the polling interval is greater than zero (0), the application files are reloaded after the application is updated. For JavaServer Pages (JSP) files in a Web module, a Web container reloads JSP files only when the IBM extension jspReloadingEnabled in the jspAttributes

of the ibm-web-ext.xmi file is set to true. You can set jspReloadingEnabled to true when editing your Web module's extended deployment descriptors in an assembly tool.

4. Change or add the following components or modules as needed:
   - Application files
   - WAR files
   - EJB Jar files
   - HTTP plug-in configuration files
5. For changes to take effect, you might need to start, stop, or restart an application. "Starting or stopping applications" on page 77 provides information on using the administrative console to start, stop, or restart an application. Starting applications with scripting and Stopping applications with scripting provide information on using the wsadmin scripting tool.

Changing or adding application files

You can change or add application files on application servers without having to stop the server and start it again.

There are several changes that you can make to deployed application files without stopping the server and starting it again.

Important: See "Ways to update application files" on page 81 and determine whether hot deployment is the appropriate way for you to update your application files. Other ways are easier and hot deployment is appropriate only for experienced users. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server.

This topic describes how to make the following changes by manipulating an application file on the server where the application is deployed:

- Updating an existing application on a running server, providing a new enterprise application (EAR file)
- Adding a new application to a running server
- Removing an existing application from a running server
- Changing or adding files to existing enterprise bean (EJB) or Web modules
- Changing the application.xml file for an application
- Changing the ibm-app-ext.xmi file for an application
- Changing the ibm-app-bnd.xmi file for an application
- Changing a non-module Jar file contained in the EAR file

Updating an existing application on a running server (providing a new EAR file)

Reinstall an updated application using the administrative console or the wsadmin $AdminApp install command with the -update option.

Both reinstallation methods enable you to update an existing application using any of the other steps listed in this file, including changing classes, adding modules, removing modules, changing modules, or changing metadata files. The application reinstallation methods detect the changes in your application and prompt you for additional binding data that might be needed to install the application. The reinstallation process automatically stops and restarts your application on the appropriate servers.

Hot deployment	Yes
Dynamic reloading	Yes

Adding a new application to a running server

Install an application using the administrative console or the wsadmin install command.

Hot deployment	Yes
Dynamic reloading	No

Removing an existing application from a running server

Stop the application and then uninstall it from the server. Use the administrative console to stop the application and then uninstall it. Or run the wasadmin `stopApplication` command and then the `uninstall` command.

Hot deployment	Yes
Dynamic reloading	No

Changing or adding files to existing EJB or Web modules

1. Update the application files in the *application_root* location.
2. Restart the application. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment	Yes
Dynamic reloading	No

Changing the application.xml file for an application

Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment	Not applicable
Dynamic reloading	Yes

Changing the ibm-app-ext.xmi file for an application

Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment	Not applicable
Dynamic reloading	Yes

Changing the ibm-app-bnd.xmi file for an application

Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment	Not applicable
Dynamic reloading	Yes

Changing a non-module Jar file contained in the EAR file

1. Update the non-module Jar file in the *application_root* location.
2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

   If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment	Yes
Dynamic reloading	Yes

Changing or adding WAR files

You can change Web application archives (WAR files) on application servers without having to stop the server and start it again.

There are several changes that you can make to WAR files without stopping the server and starting it again.

Important: See "Ways to update application files" on page 81 and determine whether hot deployment is the appropriate way for you to update your WAR files. Other ways are easier and hot deployment is appropriate only for experienced users. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server.

This topic describes how to make the following changes by manipulating a WAR file on the server where the application is deployed:

- Changing an existing JavaServer Pages (JSP) file
- Adding a new JSP file to an existing application
- Changing an existing servlet class (editing and recompiling)
- Changing a dependent class of an existing servlet class
- Adding a new servlet using the Invoker (Serve Servlets by class name) facility or adding a dependent class to an existing application
- Adding a new servlet, including a new definition of the servlet in the `web.xml` deployment descriptor for the application
- Changing the `web.xml` file of a WAR file
- Changing the `ibm-web-ext.xmi` file of a WAR file
- Changing the `ibm-web-bnd.xmi` file of a WAR file

Changing an existing JSP file

Place the changed JSP file directly in the *application_root*/*module_name* directory or the appropriate subdirectory. The change will be automatically detected and the JSP will be recompiled and reloaded.

Hot deployment	Not applicable
Dynamic reloading	Yes

Adding a new JSP file to an existing application

Place the new JSP file directly in the *application_root*/*module_name* directory or the appropriate subdirectory. The new file will be automatically detected and compiled on the first request to the page.

Hot deployment	Yes
Dynamic reloading	Yes

Changing an existing servlet class (editing and recompiling)

1. Place the new version of the servlet `.class` file directly in the *application_root*/*module_name*/`WEB-INF/classes` directory. If the `.class` file is part of a Jar file, you can place the new version of the Jar file directly in *application_root*/*module_name*/`WEB-INF/lib`. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.
2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

   If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment	Not applicable
Dynamic reloading	Yes

Changing a dependent class of an existing servlet class

1. Place the new version of the dependent .class file directly in the *application_root*/*module_name*/WEB-INF/classes directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root*/*module_name*/WEB-INF/lib. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.
2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

   If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment	Not applicable
Dynamic reloading	Yes

Adding a new servlet using the Invoker (Serve Servlets by class name) facility or adding a dependent class to an existing application

1. Place the new .class file directly in the *application_root*/*module_name*/WEB-INF/classes directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root*/*module_name*/WEB-INF/lib. In either case, the change will be detected, the Web application will be shut down and reinitialized, picking up the new class.

   This case is treated the same as changing an existing class. The difference is that adding the servlet or class does not immediately cause the Web application to reload because the class has never been loaded before. The class simply becomes available for execution.
2. If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

   If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

Hot deployment	Yes
Dynamic reloading	Not applicable

Adding a new servlet, including a new definition of the servlet in the web.xml deployment descriptor for the application

1. Place the new .class file directly in the *application_root*/*module_name*/WEB-INF/classes directory. If the .class file is part of a Jar file, you can place the new version of the Jar file directly in *application_root*/*module_name*/WEB-INF/lib.

   You can edit the web.xml file in place or copy it into the *application_root*/*module_name*/WEB-INF/classes directory. The new .class file will not trigger a reloading of the application.
2. Restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands. After the application restarts, the new servlet is available for service.

Hot deployment	Yes
Dynamic reloading	Not applicable

Changing the web.xml file of a WAR file

1. Edit the web.xml file in place or copy it into the *metadata_root*/*module_name*/WEB-INF directory.
2. Restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment	Yes
Dynamic reloading	Yes

Changing the ibm-web-ext.xmi file of a WAR file

Edit the extension settings as needed. You can change all of the extension settings. The only warning is if you set the reloadInterval property to zero (0) or the reloadEnabled property to false, the application no longer automatically detects changes to class files. Both of these changes disable the automatic reloading function. The only way to re-enable automatic reloading is to change the appropriate property and restart the application. See other task descriptions in this file for information on restarting an application.

Hot deployment	Not applicable
Dynamic reloading	Yes

Changing the ibm-web-bnd.xmi file of a WAR file

1. Edit the bindings as needed. You can change all of the values but ensure that the entities you are binding to are present in the configuration of the server.
2. Restart the application. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment	Not applicable
Dynamic reloading	Yes

Changing or adding EJB Jar files

You can change enterprise bean (EJB) Jar files on application servers without having to stop the server and start it again.

There are several changes that you can make to EJB Jar files without stopping the server and starting it again.

Important: See "Ways to update application files" on page 81 and determine whether hot deployment is the appropriate way for you to update your EJB Jar files. Other ways are easier and hot deployment is appropriate only for experienced users. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server.

This topic describes how to make the following changes by manipulating an EJB file on the server where the application is deployed:

- Changing the ejb-jar.xml file of an EJB Jar file
- Changing the ibm-ejb-jar-ext.xmi or ibm-ejb-jar-bnd.xmi file of an EJB Jar file
- Changing the Table.ddl file for an EJB Jar file
- Changing the Map.mapxmi or Schema.dbxmi file for an EJB Jar file
- Updating the implementation class for an EJB file or a dependent class of the implementation class for an EJB file
- Updating the Home/Remote interface class for an EJB file
- Adding a new EJB file to an existing EJB Jar file

Changing the ejb-jar.xml file of an EJB Jar file

Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment	Not applicable
Dynamic reloading	Yes

Change the ibm-ejb-jar-ext.xmi or ibm-ejb-jar-bnd.xmi file of an EJB Jar file

Restart the application. Automatic reloading will not detect the change. Use the administrative console to restart the application. Or run the wasadmin stopApplication and startApplication commands.

Hot deployment Not applicable
Dynamic reloading Yes

Changing the Table.ddl file for an EJB Jar file

Rerun the DDL file on the user database server. Changing the `Table.ddl` file has no effect on the application server and is a change to the database table schema for the EJB files.

Hot deployment Not applicable
Dynamic reloading Not applicable

Changing the Map.mapxmi or Schema.dbxmi file for an EJB Jar file

1. Change the `Map.mapxmi` or `Schema.dbxmi` file for an EJB Jar file.
2. Regenerate the deployed code artifacts for the EJB file.
3. Apply the new EJB Jar file to the server.
4. Restart the application. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment Not applicable
Dynamic reloading Yes

Updating the implementation class for an EJB file or a dependent class of the implementation class for an EJB file

1. Update the class file in the *application_root*/*module_name*`.jar` file.
2. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

   If automatic reloading is not enabled, restart the application of which the EJB file is a member. If the updated module is used by other modules in other applications, restart those applications as well. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment Not applicable
Dynamic reloading Yes

Updating the Home/Remote interface class for an EJB file

1. Update the interface class of the EJB file.
2. Regenerate the deployed code artifacts for the EJB file.
3. Apply the new EJB Jar file to the server.
4. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

   If automatic reloading is not enabled, restart the application of which the EJB file is a member. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment Not applicable
Dynamic reloading Yes

Adding a new EJB file to an existing EJB Jar file

1. Apply the new or updated Jar file to the *application_root* location.
2. If automatic reloading is enabled, you do not need to take further action. Automatic reloading will detect the change.

If automatic reloading is not enabled, restart the application. Use the administrative console to restart the application. Or run the wasadmin `stopApplication` and `startApplication` commands.

Hot deployment	Yes
Dynamic reloading	Yes

Changing the HTTP plug-in configuration

You can change the HTTP plug-in configuration without having to stop the server and start it again.

There are several change that you can make to the HTTP plug-in configuration without stopping the server and starting it again.

Important: See "Ways to update application files" on page 81 and determine whether hot deployment is the appropriate way for you to update your HTTP plug-in configuration. Other ways are easier and hot deployment is appropriate only for experienced users.

This file describes--

- Changing the `application.xml` file to change the context root of a Web application archive (WAR file)
- Changing the `web.xml` file to add, remove, or modify a servlet mapping
- Changing the `server.xml` file to add, remove, or modify an HTTP transport or changing the `virtualhost.xml` file to add or remove a virtual host or to add, remove, or modify a virtual host alias

Changing the application.xml file to change the context root of a WAR file

1. Change the `application.xml` file.
2. If the plug-in configuration property Automatically propagate plug-in configuration file is selected for this plug-in, it is automatically regenerated whenever the `application.xml` file changes. (See Web server plug-in properties settings for information on how to set this property.) You can also run the `GenPluginCfg.bat/sh` script, or issue a wsadmin command to regenerate the plug-in configuration file.

Hot deployment	Yes
Dynamic reloading	No

Changing the web.xml file to add, remove, or modify a servlet mapping

1. Change the `web.xml` file.
2. If the plug-in configuration property Automatically propagate plug-in configuration file is selected for this plug-in, it is automatically regenerated whenever the `web.xml` file changes. (See Web server plug-in properties settings for information on how to set this property.) You can also run the `GenPluginCfg.bat/sh` script, or issue a wsadmin command to regenerate the plug-in configuration file.

   If the Web application has file serving enabled or has a servlet mapping of /, the plug-in configuration does not have to be regenerated. In all other cases a regeneration is required.

Hot deployment	Yes
Dynamic reloading	Yes

Changing the server.xml file to add, remove, or modify an HTTP transport or changing the virtualhost.xml file to add or remove a virtual host or to add, remove, or modify a virtual host alias

1. Change the `server.xml` file to add, remove, or modify an HTTP transport or change the `virtualhost.xml` file to add or remove a virtual host or to add, remove, or modify a virtual host alias.
2. If the plug-in configuration property **Automatically propagate plug-in configuration file** is selected for this plug-in, it is automatically regenerated whenever the `server.xml` file changes. (See Web server plug-in properties settings for information on how to set this property.) You can also run the `GenPluginCfg.bat/sh` script, or issue a wsadmin command to regenerate the plug-in configuration file.

Hot deployment	Yes

Dynamic reloading Yes

Uninstalling applications

After an application no longer is needed, you can uninstall it.

Uninstalling an application deletes the application from the WebSphere Application Server configuration repository and it deletes the application binaries from the file system of all nodes where the application modules are installed.

1. Click **Applications > Enterprise Applications** in the administrative console navigation tree to access the Enterprise Applications page.
2. If you need to retain a copy of the application, back up the application.
   a. Select the application you want uninstalled.
   b. Click **Export**.

   The application is exported to an enterprise application (`.ear` file), preserving the binding information.
3. Uninstall the application.
   a. Select the application you want uninstalled.
   b. Click **Uninstall**.
4. Save changes made to the administrative configuration.

In the single-server product, application binaries are deleted after you save the changes.

Removing a file

After a file is no longer needed, you can remove the file from an application or module deployed on a server.

Removing a file deletes the file from the WebSphere Application Server configuration repository and it deletes the file from the file system of all nodes where the file is installed.

- Remove a file from an application.
  1. Go to the Enterprise Applications page. Click **Applications > Enterprise Applications** in the console navigation tree.
  2. Select the application that contains a file you want removed.
  3. Click **Remove File**. The Remove a file page is displayed
  4. Select the URI of the file that you want removed from the application.
  5. Back up the application.

     Under **Export before removing file**, select the application name and then specify the location to which you want the file exported.
  6. Click **OK** to remove the file.
- Remove a file from a module.
  1. Go to the Manage modules page.

     Click **Applications > Enterprise Applications >** *application_name* **> Manage modules** in the console navigation tree.
  2. Select the module from which you want to delete a file.
  3. Click **Remove File**. The Remove a file from a module page is displayed.
  4. Select the URI of the file that you want removed from the module.
  5. Back up the application.

Under **Export before removing file**, select the application name and then specify the location to which you want the file exported.

6. Click **OK** to remove the file.

The file is exported to the designated location and removed from the application or module. The application or standalone Web module that had a file removed is restarted so the changes take effect.

Save the changes to your administrative configuration. In the single-server product, application binaries are deleted after you save the changes.

Common deployment framework

The *common deployment framework* enables you to implement plug-ins that add steps to default Java 2 Platform, Enterprise Edition (J2EE) application management operations such as install, uninstall, edit and update.

Using the framework, you can implement management operations on specific types of deployable contents. For example, the deployable contents might include EAR, WAR, JAR or other J2EE modules and the management operations might include install and uninstall. Each operation is divided into a number of steps. For example, the install operation has steps for EJBDeploy and JavaServer Pages (JSP) compilation, among others. Using the common deployment framework, you can add steps to the default logic for J2EE operations.

Version 6.1 supports framework plug-ins that extend deployment of EAR files. An EAR file has operations such as createEarWrapper, installApplication, uninstallApplication and editApplication. Using a framework plug-in, you can add steps to default install operations that support, for example, creating additional configuration artifacts in a configuration session, modifying an input EAR file using code generation, or additional validating of input parameters.

To extend application management operations using the framework, a plug-in must do the following:

- Implement each step.

  A *step* runs logic that performs an operation. A step can access the deployment context and the deployable object. The *deployment context* provides information such as the operation name, the configuration session identifier, the temporary location for creating temporary files, operations parameters, and the like. A step is added by the extension provider.
- Implement an extension provider that adds each implemented step.

  An *extension provider* is a class that provides steps for an operation on a given type. For Version 6.1, it is the EAR file type.
- Register the plug-in with a WebSphere Application Server server.

  The plug-in is implemented as an Eclipse plug-in and is placed in *app_server_root*/`plugins` directory. Add the extension point for the extension provider in the `META-INF/plugin.xml` file within the plug-in JAR file.

For an example of these steps, refer to Extending application management operations through programming.

Deploying and administering applications: Resources for learning

Use the following links to find relevant supplemental information about deploying and administering applications using the administrative console. 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.

Refer to Web resources for learning for links to information applicable to WebSphere Application Server generally, such as lists of IBM technical papers, Redbooks and samples.

View links to additional information about:
- “Programming model and decisions”
- “Programming instructions and examples”
- “Administration”

Programming model and decisions
- Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition, Second Edition
- The J2EE™ Tutorial
- Building Java™ Enterprise Applications Volume I: Architecture
- Recommended reading list: J2EE and WebSphere Application Server

Programming instructions and examples
- *IBM WebSphere: Deployment and Advanced Configuration*, Roland Barcia, *et al.*, ISBN 0131468626 (Prentice Hall, 2004)
- WebSphere Application Server - Express V6 Developers Guide and Development Examples
- WebSphere Application Server - Express: Pathways to Success on the Web
- *IBM WebSphere Developer Technical Journal:* Co-hosting multiple versions of J2EE applications
- Automated Deployment of Enterprise Application Updates: Part 1 - Basic concepts
- *IBM WebSphere Developer Technical Journal:* The top 10 (more or less) J2EE best practices

Administration
- *IBM WebSphere Developer Technical Journal:* System management for WebSphere Application Server V6 -- Part 1 Overview of system management enhancements
- *IBM WebSphere Developer Technical Journal:* System management for WebSphere Application Server V6 -- Part 5: Flexible options for updating deployed applications
- WebSphere Application Server V6 System Management & Configuration Handbook
- WebSphere Application Server V6 Migration Guide
- WebSphere Version 6 Web Services Handbook Development and Deployment
- Listing of all IBM WebSphere Application Server Redbooks

Chapter 7. Web applications

Task overview: Developing and deploying Web applications

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

1. **(Optional)** Migrate existing Web applications to run in the new version of WebSphere Application Server.
2. Design the Web application and develop its code artifacts: Servlets, JavaServer Pages (JSP) files, and static files, as for example, images and Hyper Text Markup Language (HTML) files. See the "Web applications: Resources for learning" on page 138 topic for links to design documentation.

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

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

Web applications

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

The files in a Web application are related in that they work together to perform a business logic function. For example, one of the WebSphere Application Server samples is a *Simple Greeting* Web application. This application, comprised of a servlet and Web pages, greets new users when they access the application.

The Web application is a concept supported by the Java Servlet Specification. Web applications are typically packaged as `.war` files.

web.xml file

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

The Java Servlet 2.4 specification defines the web.xml deployment descriptor file in terms of an XML schema document. For backwards compatibility of applications written to the Java Servlet 2.2 Specification, Web containers are also required to support the Java Servlet 2.2 specification. For backwards compatibility of applications written to the Java Servlet 2.3 specification, Web containers are also required to support the Java Servlet 2.3 specification.

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

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

Location

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

For example, if the application is client.war, then the web.xml file is placed in the *install_root*/*client war*/`WEB-INF` directory.

Usage notes

- Is this file read-only?

  No
- Is this file updated by a product component?

  This file is updated by the Application Server Toolkit.
- If so, what triggers its update?

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

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

Sample file entry

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

```
  <url-patter>ForwardedServlet</url-pattern>
  <dispatcher>FORWARD</dispatcher>
 </filter-mapping>
 <listener>
  <listener-class>tests.ContextListener</listener-class>
 </listener>
 <listener>
  <listener-class>tests.ServletRequestListener.RequestListener</listener-class>
 </listener>
 <servlet>
  <servlet-name>welcome</servlet-name>
  <servlet-class>WelcomeServlet</servlet-class>
 </servlet>
 <servlet>
  <servlet-name>ServletErrorPage</servlet-name>
  <servlet-class>tests.Error.ServletErrorPage</servlet-class>
 </servlet>
 <servlet>
  <servlet-name>IncludedServlet</servlet-name>
  <servlet-class>tests.Filter.IncludedServlet</servlet-class>
 </servlet>
 <servlet>
  <servlet-name>ForwardedServlet</servlet-name>
  <servlet-class>tests.Filter.ForwardedServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>welcome</servlet-name>
  <url-pattern>/hello.welcome</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
  <servlet-name>ServletErrorPage</servlet-name>
  <url-pattern>/ServletErrorPage</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
  <servlet-name>IncludedServlet</servlet-name>
  <url-pattern>/IncludedServlet</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
  <servlet-name>ForwardedServlet</servlet-name>
  <url-pattern>/ForwardedServlet</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
  <welcome-file>hello.welcome</welcome-file>
 </welcome-file-list>
 <error-page>
  <exception-type>java.lang.ArrayIndexOutOfBoundsException</exception-type>
  <location>/ServletErrorPage</location>
 </error-page>
</web-app>
```

Default Application

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

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

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

Snoop servlet

Use the Snoop servlet to retrieve information about a servlet request. This servlet returns the following information:
- Servlet initialization parameters
- Servlet context initialization parameters
- URL invocation request parameters
- Preferred client locale
- Context path
- User principal
- Request headers and their values
- Request parameter names and their values
- HTTPS protocol information
- Servlet request attributes and their values
- HTTP session information
- Session attributes and their values

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

The URL for the Snoop servlet is: `http://localhost:9080/snoop/`.

HelloHTML servlet

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

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

transition: The PageList Servlet custom extension is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. Re-architect your legacy applications to use javax.servlet.filter classes instead of com.ibm.servlet classes. Starting from the Servlet 2.3 specification, javax.servlet.filter classes you can intercept requests and examine responses. You can also use javax.servlet.filter classes to achieve chaining functionality, as well as embellishing or truncating responses.

HitCount application

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

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

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

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

The URL for the HitCount application is: `http://localhost:9080/HitCount.jsp`.

Servlets

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

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

For the purposes of WebSphere Application Server, discussions of servlets focus on Hyper Text Transfer Protocol (HTTP) servlets, which serve Web-based clients.

With the introduction of Java Servlet 2.4 specification, you can define servlets as welcome files. Non-servlet resources are served only when the FileServingEnabled attribute is set to true. Serving welcome files is connected to serving static content, therefore fileServing enabled is set in the Web module.

JavaServer Pages

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

JSP files support a division of roles:

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

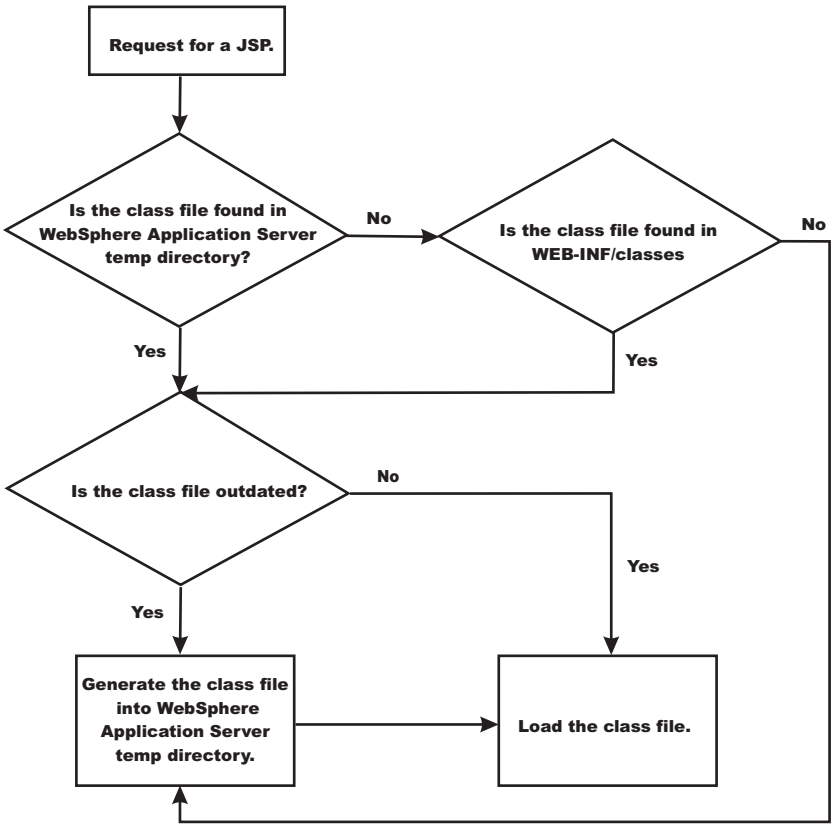
Java programmers
: Create the reusable Java components and provide the HTML authors with the component names and attributes.

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

WebSphere Application Server Version 6.1 supports the JSP 2.0 specification. The sub-topics below discuss WebSphere Application Server's JSP 2.0 implementation, focusing on configuration, tools and extensions.

JSP class file generation

At runtime, the WebSphere Application Server JavaServer Pages (JSP) engine loads JSP class files from either the WebSphere Application Server `temp` directory or a Web module's `WEB-INF/classes` directory. The WebSphere Application Server `temp` directory is typically *`WAS_INSTALL_ROOT`*`/AppServer/profiles/default/temp/`*`node_name`*`/`*`server_name`*. The JSP engine first searches for a class file in the `temp` directory and then it searches in the Web module's `WEB-INF/classes` directory. Figure 1 shows the processing logic of the JSP engine at runtime.



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

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

Packages and directories for generated .java and .class files

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

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

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

The usage of full package names enables the configuration of a JSP as a servlet in the `web.xml` file. See "JSP class loading" on page 108 for more information. The table below gives examples of packages and directory structures for generated .java and .class files.

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

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

Dependency information

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

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

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

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

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

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

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

Version, JSP engine options, and WEB.XML information

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

- The date and time the `.java` file was generated

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

```
/*
profile_root/AppSrv01/installedApps/MyCell/sampleApp.ear/examples.war/WEB-INF/classes/_ibmjsp/_jsp1.java
was generated @ Wed May 03 10:05:56 EDT 2006IBM WebSphere Application Server - ND, 6.1.0.0
    Build Number: o0441.04
    Build Date: 05/01/06*******************************************************
The JSP engine configuration parameters were set as follows:

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



outputDir =                [C:/WebSphere_6.0/AppServer/profiles/AppSrv01/installedApps/MyCell/
                             sampleApp.ear/examples.war/WEB-INF/classes]
reloadEnabled =            [true]
reloadEnabledSet =         [true]
reloadInterval =           [5000]
trackDependencies =        [false]
usePageTagPool =           [false]
useThreadTagPool =         [true]
useImplicitTagLibs =       [true]
verbose =                  [false]
looseLibMap =              [null]
useJikes =                 [false]
useFullPackageNames =      [true]
translationContextClass =  [null]
extensionProcessorClass =  [null]
javaEncoding =             [UTF-8]
autoResponseEncoding =     [false]

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

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

JSP class loading

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

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

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

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

Configuring JSP files as Servlets

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

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

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

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

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

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

- An example of a configuration repository directory is {WAS_ROOT}/profiles/profilename/config/cells/*cellname*/applications/*enterpriseappname*/deployments/*deployedname*/*webmodulename*

- An example of an application binaries directory is: {WAS_ROOT}/profiles/*profilename*/installedApps/*nodename*/EnterpriseAppName/*WebModuleName*/

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

Configuring JSP run time reloading

JSP files can be translated and compiled at run time when the JSP file or its dependencies are modified. This is known as JSP reloading. JSP reloading is enabled through the **reloadEnabled** JSP engine parameter in the WEB-INF/ibm-web-ext.xmi file:

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

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

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

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

Reload interval

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

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

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

Dependency tracking

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

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

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

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

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

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

Disabling compilation

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

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

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

Reload processing sequence

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

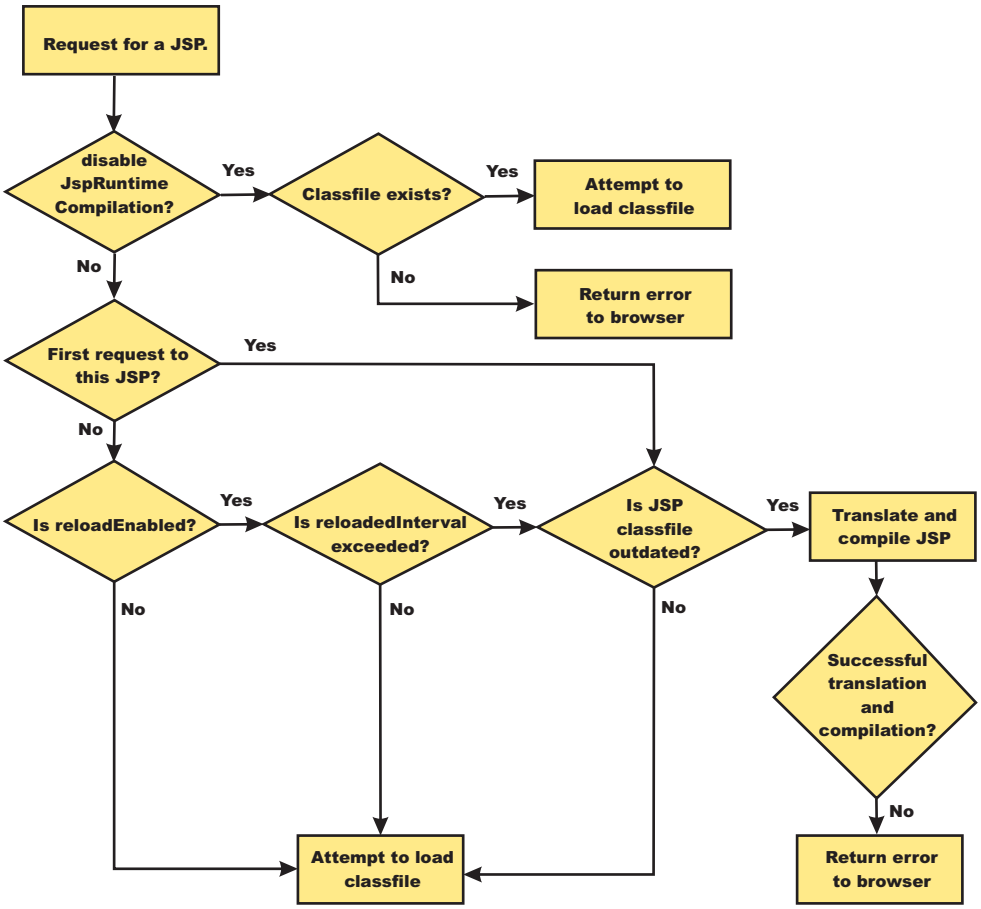


Figure 1. Reload processing sequence when **trackDependencies** *is false.*

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

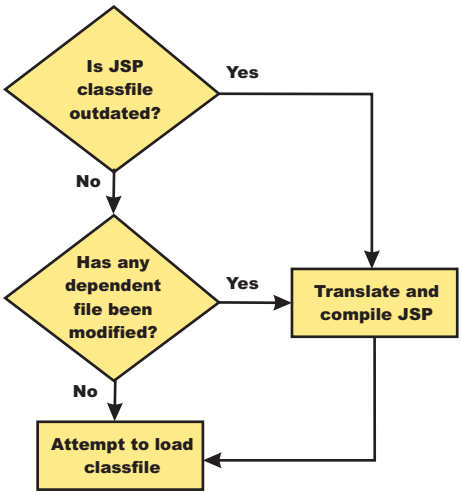


Figure 2. Additional reload processing performed when **trackDependencies** *is true.*

JSP reload options for Web modules settings

Use this panel to configure the class reloading of Web modules such as JavaServer Pages (JSP) files

To view this administrative console panel, click **Applications > Enterprise Applications >** *application_name* **> JSP reload options for Web modules**. This panel is the same as the **Provide JSP reloading options for Web modules** panel on the application installation and update wizards.

Web module:

Specifies the name of a JSP file in the application.

URI:

Specifies the location of the module 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 jspReloadingEnabled in the jspAttributes of the `ibm-web-ext.xmi` file is set to `true`.

JSP reload interval in seconds:

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

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.

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

Disabling JavaServer Pages run time compilation

By default, the JavaServer Pages (JSP) engine translates a requested JSP file, compiles the `.java` file, and loads the compiled servlet into the run time environment. You can change the JSP engine default behaviour by indicating a JSP file should never be translated or compiled at run time, even when a `.class` file does not exist.

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

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

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

- To disable the translation and compilation of JSP files for all Web applications, set the Web container custom property `disableJspRuntimeCompilation` to `true`.

  Set this property through the Web container `Custom properties` panel in the administrative console. To view this administrative console page, click:

  Servers > **Application servers** > *server_name* > **Web container settings** > **Web container** > **Custom properties** > *property_name*

  Valid values for this setting are `true` or `false`. If this property is set to `true`, then translation and compilation of the JSP files is disabled at run time for all Web applications.
- To disable the translation and compilation of JSP files for a specific Web application, set the JSP engine initialization parameter `disableJspRuntimeCompilation` to `true`. This setting, if enabled, determines the run time behavior of the JSP engine and overrides the Web container custom property setting.

  Set this parameter through the `JavaServer Pages attribute assembly settings` panel in the Assembling applications.

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

If you disable run time compilation and a request arrives for a JSP file that does not have a matching `.class` file, the JSP engine returns HTTP error 500 (Internal server error) to the browser. In this case, an exception is written to the System Out (SYSOUT) and First Failure Data Capture (FFDC) logs.

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

Provide options to compile JavaServer Pages settings

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

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

You can specify the JSP compiler options on this panel 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's `WEB-INF/ibm-web-ext.xmi` file to change its JSP compiler options.

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`, which specifies source level 1.3.

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.

JSP batch compilation

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

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

The JSP batch compiler works on Web modules that support Servlet 2.2 and up through Servlet 2.4 The batch compiler works on JSP pages written to the JSP 2.0 specification or previous specifications back to JSP 1.0. It recognizes a Servlet 2.4 deployment descriptor, `web.xml`, and can use any jsp-config elements that it may contain. In a Servlet 2.3 (JSP 1.2) or Servlet 2.2 (JSP 1.1) deployment descriptor the batch compiler recognizes and uses any taglib elements that the descriptor may contain.

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

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

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

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

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

Processing of Web modules

The batch compiler operates on one Web module at a time. If the target is either an EAR file or an installed enterprise application that contains more than one Web module, the batch compiler operates on each Web module individually. This is done because JSP pages are configured on a Web module basis,

through the Web module's web.xml deployment descriptor file. Within a Web module, the batch compiler processes one directory at a time. It validates and translates each JSP page individually, and then invokes the Java compiler for the entire group of generated Java sources files in that directory. If one JSP page fails during the Java compilation phase, the Java compiler might not create class files for most or all of the JSP pages that successfully compiled in that directory.

JSP file extensions

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

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

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

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

Batch compiler command

Both a Windows batch file, `JspBatchCompiler.bat` and UNIX shell script `JspBatchCompiler.sh` for running the batch compiler from the command line are found in the `{WAS_ROOT}/bin` directory. An Ant task is also available for executing the batch compiler using Ant. See the topic, Batch Compiler Ant Task for additional information.

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

```
JspBatchCompiler -ear.path | -war.path | -enterpriseapp.name <name>
   [-response.file <filename>]
   [-webmodule.name <name>]
   [-filename <jsp name | directory name>
   [-recurse <true | false>]
   [-config.root <path>]
   [-cell.name <name>]
   [-node.name <name>]
   [-server.name <name>]
   [-profileName <name>]
   [-extractToDir <path>]
   [-compileToDir <path>]
   [-compileToWebInf <true | false>]
   [-translate <true | false>]
   [-compile <true | false>]
   [-removeTempDir <true | false>]
   [-forceCompilation <true | false>]
   [-useFullPackageNames <true | false>]
```

```
[-trackDependencies <true | false>]
[-createDebugClassfiles <true | false>]
[-keepgenerated <true | false>]
[-keepGeneratedclassfiles <true | false>]
[-usePageTagPool <true | false>]
[-useThreadTagPool <true | false>]
[-classloader.parentFirst <true | false>]
[-classloader.singleWarClassloader <true | false>]
[-additional.classpath <classpath to additional JAR files and classes>]

[-verbose <true | false>]
[-deprecation <true | false>]
[-javaEncoding <encoding>
[-jdkSourceLevel <13 | 14 | 15>]
[-compilerOptions <space-separated list of java compiler options>]
[-useJikes <true | false>]
[-jsp.file.extensions <file extensions to process>]
[-log.level <SEVERE | WARNING | INFO | CONFIG | FINE | FINER | FINEST | OFF>]
*** See batchcompiler.properties.default in WAS_ROOT/bin for more information. ***
*** See JspCBuild.xml in WAS_ROOT/bin for information about the public WebSphere Ant task JspC. ***
```

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

1. JSP engine configuration parameters for a Web module.

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

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

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

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

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

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

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

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

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

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

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

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

  In a default installation, the template response file, batchcompiler.properties.default, is found in the `{WAS_ROOT}/bin` directory. Copy this template to create your own response files containing defaults for the parameters in which you are interested. All the required and optional parameters (except response.file) can be configured in a response file. **For example:** JspBatchCompiler -response.file c:\myproject\batchc.props

  Default : null
- **webmodule.name**

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

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

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

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

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

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

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

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

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

  Default: true; All directories beneath the target directory are processed.
- **config.root**

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

  Default: `{WAS_ROOT}/profiles/profilename/config`
- **cell.name**

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

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

- **node.name**

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

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

- **server.name**

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

  Default: server1

- **profileName**

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

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

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

- **extractToDir**

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -extractToDir c:\myTempDir.

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

  Default values:

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

  The default is `server1`.

- **compileToDir**

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

  Example:: JspBatchCompiler -enterpriseApp.name sampleApp -compileToDir c:\myTargetDir

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

Default values:

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

- **compileToWebInf**

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

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

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

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

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

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -forceCompilation true.

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

  Default: false
- **useFullPackageNames**

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

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

  Example: JspBatchCompiler –enterpriseApp.name sampleApp –useFullPackageNames true

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

Default: false

- **removeTempDir**

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

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

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

  Default: false

- **translate**

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

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

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

  Default: true

- **compile**

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

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -compile false

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

  Default: true

- **trackDependencies**

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

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -trackDependencies true.

  Use-case: Useful in a development environment.

  Default: false

- **createDebugClassfiles**

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

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -createDebugClassfiles true

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

  Default: false

- **keepgenerated**

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

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

Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -keepgenerated true

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

Default: false

- **keepGeneratedclassfiles**

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -keepGeneratedclassfiles false -keepgenerated false

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

  Default: true

- **usePageTagPool**

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -usePageTagPool true

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

  Default: false

- **useThreadTagPool**

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -useThreadTagPool true

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

  Default: false

- **classloader.parentFirst**

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -classloader.parentFirst false

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

  Default: true

- **classloader.singleWarClassloader**

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -classloader.singleWarClassloader true

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

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

- **additional.classpath**

  Specifies additional class path entries to be used when parsing and compiling JSP pages. This parameter is used only when `war.path` is given. When `war.path` is the target, WebSphere Shared Libraries are not picked up by the batch compiler. Therefore, if your WAR file relies on, for example, a JAR file that is configured in WebSphere Application Server as a shared library, then use this option to point to that JAR file. In addition, if you give `war.path` and also use the `-extractToDir` parameter, then any JAR files that are in the WAR file's manifest class-path is not added to the class path (since the

WAR file has now been extracted by itself outside the EAR file in which it resides). Use `-additional.classpath` in this case to point to the necessary JAR files. Add the full path to needed resources, separated by your system-dependent path separator.

Example: JspBatchCompiler -war.path c:\myproject\examples.war -additional.classpath c:\myJars\someJar.jar;c:\myClasses

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

Default: null

- **verbose**

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

  Example: JspBatchCompiler -war.path c:\myproject\examples.war -verbose true

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

  Default: false

- **deprecation**

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

  Example: JspBatchCompiler -war.path c:\myproject\examples.war -deprecation true

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

  Default: false

- **javaEncoding**

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

  Example: JspBatchCompiler -war.path c:\myproject\examples.war -javaEncoding Shift-JIS

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

  Default value: UTF-8.

- **jdkSourceLevel**

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

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

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

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

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

  Default value: 13

- **compilerOptions**

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

  Example: JspBatchCompiler -war.path c:\myproject\examples.war -compilerOptions ″ -bootclasspath <path>″

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

Default: null

- **useJikes**

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

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -useJikes true

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

  Default value: false

- **jsp.file.extensions**

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

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

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

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

- **log.level**

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

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

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

  Default: CONFIG

Batch compiler ant task:

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

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

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

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

On Windows:

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

On UNIX or i5/OS:

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

Example build.xml Using the JspC Task

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

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

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

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

Batch compiler class path:

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

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

Global tag libraries

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

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

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

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

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

tsx:dbconnect tag JavaServer Pages syntax (deprecated):

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

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

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

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

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

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

where:

- **id**

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

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

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

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

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

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

The application server supports connection to JDBC databases and ODBC databases.

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

```
url="jdbc:db2:sample"
driver="com.ibm.db2.jdbc.app.DB2Driver"
```

– For an ODBC database, use the Sun JDBC-to-ODBC bridge driver included in their Java2 Software Developers Kit (SDK) or another vendor's ODBC driver.

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

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

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

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

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

- **jndiname**

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

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

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

dbquery tag JavaServer Pages syntax (deprecated):

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

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

The <tsx:dbquery> tag does the following:

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

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

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

where:

- **id**

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

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

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

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

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

- **connection**

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

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

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

dbmodify tag JavaServer Pages syntax (deprecated):

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

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

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

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

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

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

where:
- **connection**

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

- Database commands

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

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

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

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

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

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

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

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

where:

- **name**

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

- **property**

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

Tag example:

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

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

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

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

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

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

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

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

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

where:

- **<tsx:getProperty>**

  Represents the syntax as a mechanism for embedding variable data.
- **userid**

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

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

tsx:repeat tag JavaServer Pages syntax (deprecated):

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

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

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

- The end value is reached.
- An exception is thrown.

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

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

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

where:

- **index**

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

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

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

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

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

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

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

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

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

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

Example: Using tsx:repeat JavaServer Pages tag to iterate over a results set (deprecated): Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

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

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

Some observations about the bean:

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

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

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

Implicit and explicit indexing

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

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

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

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

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

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

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

Nesting <tsx:repeat> blocks

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

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

JavaServer Pages migration best practices and considerations

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

Classes from the unnamed or default package

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

Page encoding for JSP documents

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

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

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

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

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

web.xml file version

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

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

jsp:useBean tag

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

Generated packages for JSP classes

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

JspServlet class

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

Web modules

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

A Web module contains:

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

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

You can create Web modules as stand-alone applications, or you can combine Web modules with other modules to create Java 2 Platform, Enterprise Edition (J2EE) applications. You install and run a Web module in the Web container of an application server.

Troubleshooting tips for Web application deployment

Deployment of a Web application is successful if you can access the application by typing a Uniform Resource Locator (URL) in a browser, or if you can access the application by following a link.

If you cannot access your application, follow these steps to eliminate some common errors that can occur during migration or deployment.

Web module does not run in WebSphere Application Server Version 5.x or 6.x

Symptom Your Web module does not run when you migrate it to Version 5.x or 6.x

Problem In Version 4.x, the classpath setting that affected visibility was *Module Visibility Mode*. In Versions 5.x and 6.x, you must use class loader policies to set visibility.

Recommended response Reassemble an existing module, or change the visibility settings in the class loader policies.

See "Class loaders" on page 15 and Chapter 5, "Class loading," on page 15 for more information.

Welcome page is not visible.

Symptom You cannot access an application with a Web path of:

```
/webapp/myapp
```

Problem The default welcome page for a Web application is assumed to be *index.html*. You cannot access the default page of the *myapp* application unless it is named *index.html*.

Recommended response To identify a different welcome page, modify the properties of the Web module during assembly, see the topic, Assembling Web applications for more information in the *Developing and deploying applications* PDF book.

HTML files are not found.

Symptom Your Web application ran successfully on prior versions, but now you encounter errors that the welcome page (typically *index.html*), or referenced HTML files are not found:

```
Error 404: File not found: Banner.html
Error 404: File not found: HomeContent.html
```

Problem

For security and consistency reasons, Web application URLs are now case-sensitive on all operating systems.

Suppose the content of the index page is as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 5.0 Frameset//EN">
<HTML>
<TITLE>
Insurance Home Page
</TITLE>
    <frameset   rows="18,80">
    <frame      src="Banner.html"           name="BannerFrame"  SCROLLING=NO>
    <frame      src="HomeContent.html"      name="HomeContentFrame">
    </frameset>
</HTML>
```

However the actual file names in the `\WebSphere\AppServer\installedApps\...` directory where the application is deployed are:

```
banner.html
homecontent.html
```

Recommended response

To correct this problem, modify the *index.html* file to change the names *Banner.html* and *HomeContent.html* to *banner.html* and *homecontent.html* to match the names of the files in the deployed application.

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 IBM Support page.

Web applications: Resources for learning

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

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

Programming model and decisions

- J2EE BluePrints for Web applications
- Redbook on the design and implementation of Servlets, JSP files, and enterprise beans

Programming instructions and examples

- WebSphere Studio Application Developer Programming Guide
- Sun's Java™ Tutorial on Servlets and JavaServer Pages
- Web delivered samples in the Samples Gallery

Programming specifications

- Java 2 Software Development Kit (SDK)
- Servlet 2.4 Specification
- JavaServer Pages 2.0 Specification
- Differences between JavaScript and ECMAScript
- ISO 8859 Specifications
- Java 2 Platform, Standard Edition (J2SE)

Tuning URL invocation cache

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

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

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

To change the size of the invocation cache:

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

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

Tuning URL invocation cache

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

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

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

To change the size of the invocation cache:

1. In the administrative console, click **Servers > Application servers** and select the application server you are tuning.
2. Click **Process Definition** under Additional Properties.
3. Click **Java Virtual Machine** under Additional Properties.
4. Click **Custom Properties** under Additional Properties.
5. Specify **invocationCacheSize** in the Name field and the size of the cache in the Value field. The default size for the invocation cache is 500 entries. Since the invocation cache is no longer

thread-based, the invocation cache size specified by the user is multiplied by ten to provide similar function from previous releases. For example, if you specify an invocation cache size of 50, the Web container will create a cache size of 500.

6. Click **Apply** and then **Save** to save your changes.
7. Stop and restart the application server.

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

Task overview: Managing HTTP sessions

IBM WebSphere Application Server provides a service for managing HTTP sessions: Session Manager. The key activities for session management are summarized below.

Before you begin these steps, make sure you are familiar with the programming model for accessing HTTP session support in the applications following the Servlet 2.4 API.

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

Sessions

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

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

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

A servlet distinguishes users by their unique session IDs. The session ID arrives with each request. If the user's browser is cookie-enabled, the session ID is stored as a cookie. As an alternative, the session ID can be conveyed to the servlet by URL rewriting, in which the session ID is appended to the URL of the servlet or JavaServer Pages (JSP) file from which the user is making requests. For requests over HTTPS or Secure Sockets Layer (SSL), Another alternative is to use SSL information to identify the session.

HTTP session migration

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

Migration from Version 5.x

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

When you migrate between releases of WebSphere Application Server Version 5.x and later and you are using a database for session persistence, you can share the session database table between releases.

For example, if you are accessing applications that are on WebSphere Application Server version 5.x you can share the session id with applications running on Version 6.x.

Session security support

You can integrate HTTP sessions and security in WebSphere Application Server. When security integration is enabled in the session management facility and a session is accessed in a protected resource, you can access that session only in protected resources from then on. You cannot mix secured and unsecured resources accessing sessions when security integration is turned on. Security integration in the session management facility is not supported in form-based login with SWAM.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

Security integration rules for HTTP sessions

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

Programmatic details and scenarios

WebSphere Application Server maintains the security of individual sessions.

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

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

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

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

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

	Unauthenticated HTTP request is used to retrieve a session	HTTP request is authenticated, with an identity of ″FRED″ used to retrieve a session
A session ID for a valid session is passed in. The current session user name is BOB	The session is not returned. An UnauthorizedSessionRequestException error is created*	The session is not returned. An UnauthorizedSessionRequestException error is created*

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

Session management support

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

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

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

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

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

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

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

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

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

Session tracking options

There are several options for session tracking, depending on what sort of tracking method you want to use:

- Session tracking with cookies
- Session tracking with URL rewriting
- Session tracking with Secure Sockets Layer (SSL) information

Session tracking with cookies

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

Session tracking with URL rewriting

An application that uses URL rewriting to track sessions must adhere to certain programming guidelines. The application developer needs to do the following:

- Program servlets to encode URLs
- Supply a servlet or JavaServer Pages (JSP) file as an entry point to the application

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

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

Program session servlets to encode URLs

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

Rewrite URLs to return to the browser

Suppose you currently have this statement:

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

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

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

Rewrite URLs to redirect

Suppose you currently have the following statement:

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

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

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

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

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

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

Supply a servlet or JSP file as an entry point

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

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

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

Session tracking with SSL information

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

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

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

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

Session recovery support

For session recovery support, WebSphere Application Server provides distributed session support in the form of database sessions. Use session recovery support under the following conditions:

- When the user's session data must be maintained across a server restart
- When the user's session data is too valuable to lose through an unexpected server failure

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

Clustered session support

A clustered environment supports load balancing, where the workload is distributed among the application servers that compose the cluster. In a cluster environment, the same Web application must exist on each of the servers that can access the session. You can accomplish this setup by installing an application onto a cluster definition. Each of the servers in the group can then access the Web application

In a clustered environment, the session management facility requires an affinity mechanism so that all requests for a particular session are directed to the same application server instance in the cluster. This requirement conforms to the Servlet 2.3 specification in that multiple requests for a session cannot coexist in multiple application servers. One such solution provided by IBM WebSphere Application Server is *session affinity* in a cluster; this solution is available as part of the WebSphere Application Server plug-ins for Web servers. It also provides for better performance because the sessions are cached in memory. In clustered environments other than WebSphere Application Server clusters, you must use an affinity mechanism (for example, IBM WebSphere Edge Server affinity).

If one of the servers in the cluster fails, it is possible for the request to reroute to another server in the cluster. If distributed sessions support is enabled, the new server can access session data from the database or another WebSphere Application Server instance. You can retrieve the session data only if a new server has access to an external location from which it can retrieve the session.

Session management tuning

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

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

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

Base in-memory session pool size

The base in-memory session pool size number has different meanings, depending on session support configuration:

- With in-memory sessions, session access is optimized for up to this number of sessions.

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

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

Overflow in non-distributed sessions

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

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

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

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

Tuning parameter settings

Use this page to set tuning parameters for distributed sessions.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Web container settings > Session management > Distributed environment settings > Custom tuning parameters**.

Tuning level:

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

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

Very high (optimize for performance)

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

High

Write frequency	Time based
Write interval	300 seconds
Write contents	All session attributes

Schedule sessions cleanup	false

Medium

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

Low (optimize for failover)

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

Custom settings

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

Tuning parameter custom settings

Use this page to customize tuning parameters for distributed sessions.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* **Web container settings** > **Session management** > **Distributed environment settings** > **Custom tuning parameters** > **Custom settings**.

Write frequency:

Specifies when the session is written to the persistent store.

End of servlet service	A session writes to a database or another WebSphere Application Server instance after the servlet completes execution.
Manual update	A programmatic sync on the IBMSession object is required to write the session data to the database or another WebSphere Application Server instance.
Time based	Session data writes to the database or another WebSphere Application Server instance based on the specified Write interval value. Default: 10 seconds

Write contents:

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

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

Schedule sessions cleanup:

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

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

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

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

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

Best practices for using HTTP Sessions

best-practices: Browse the following recommendations for implementing HTTP sessions.

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

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

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

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

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

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

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

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

In compliance with the Java Servlet specification, the distributed servlet container must create an IllegalArgumentException for objects when the container cannot support the mechanism necessary for migration of the session storing them. An exception is created only when you have selected distributable.

- **The HTTPSession API does not dictate transactional behavior for sessions.**

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

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

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

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

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

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

  You can improve performance by not breaking session affinity. Some suggestions to help avoid breaking session affinity are:
  - Combine all Web applications into a single application server instance, if possible, and use modeling or cloning to provide failover support.
  - Create the session for the frame page, but do not create sessions for the pages within the frame when using multi-frame JSP files. (See discussion later in this topic.)

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

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

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

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

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

  Note: An alternative to using the manual updates is to utilize the timed updates to persist data at different time intervals. This action provides similar results as the manual update scheme.
- Implement the following suggestions to achieve high performance:
  - If your applications do not change the session data frequently, use Manual Update and the sync function (or timed interval update) to efficiently persist session information.

  - Keep the amount of data stored in the session as small as possible. With the ease of using sessions to hold data, sometimes too much data is stored in the session objects. Determine a proper balance of data storage and performance to effectively use sessions.
  - If using database sessions, use a dedicated database for the session database. Avoid using the application database. This helps to avoid contention for JDBC connections and allows for better database performance.
  - If using memory-to-memory sessions, employ partitioning (either group or single replica) as your clusters grow in size and scaling decreases.
  - Verify that you have the latest fix packs for the WebSphere Application Server.
- Utilize the following tools to help monitor session performance.
  - Run the com.ibm.servlet.personalization.sessiontracking.IBMTrackerDebug servlet. - To run this servlet, you must have the servlet invoker running in the Web application you want to run this from. Or, you can explicitly configure this servlet in the application you want to run.
  - Use the WebSphere Application Server Resource Analyzer which comes with WebSphere Application Server to monitor active sessions and statistics for the WebSphere Application Server environment.
  - Use database tracking tools such as ″Monitoring″ in DB2. (See the respective documentation for the database system used.)

HTTP session manager troubleshooting tips

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

Here are some steps to take:

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

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

  - Within this block, look for any errors or exceptions containing a package name of com.ibm.ws.webcontainer.httpsession. If none are found, this is an indication that the session manager started successfully.
  - Error ″**SRVE0054E: An error occurred while loading session context and Web application**″ indicates that SessionManager didn't start properly for a given application.
  - Look within the logs for any Session Manager related messages. These messages will be in the format SESNxxxxE and SESNxxxxW for errors and warnings, respectively, where xxxx is a number identifying the precise error. Look up the extended error definitions in the Session Manager message table.
- See the Best practices for using HTTP Sessions section in the *Developing and deploying applications* PDF books for more details.
- To dynamically view the number of sessions as a Web application is running, enable performance monitoring for HTTP sessions. This will give you an indication as to whether sessions are actually being created.
- To learn how to view the HTTP session counters as the application runs, read the Monitoring performance with Tivoli Performance Viewer chapter of the *Administering applications and their environment* PDF book.
- Alternatively, a special servlet can be invoked that displays the current configuration and statistics related to session tracking. This servlet has all the counters that are in performance monitor tool and has some additional counters.
  - Servlet name: **com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - It can be invoked from any Web module which is enabled to serve by class name. For example, using default_app, **http://localhost:9080/servlet/com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.

  - If you are viewing the module via the serve-by-class-name feature, be aware that it may be viewable by anyone who can view the application. You may wish to map a specific, secured URL to the servlet instead and disable the serve-servlets-by-classname feature.
- Enable tracing for the HTTP Session Manager component:
  - Use the trace specification **com.ibm.ws.webcontainer.httpsession.*=all=enabled**. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.
  - If you are using persistent sessions based on memory replication, also enable trace for **com.ibm.ws.drs.***.
- If you are using **database-based persistent sessions**, look for problems related to the **data source** the Session Manager relies on to keep session state information. For details on diagnosing database related problems see the Errors accessing a datasource or connection pool in the *Administering applications and their environment* PDF book

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

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

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

The application server gets EC3 - 04130007 ABENDs

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

If none of these steps fixes your problem, 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.

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 IBM Support page.

Problems creating or using HTTP sessions

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

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

What kind of problem are you having?
- HTTP Sessions are not getting created, or are lost between requests.
- HTTP Sessions are not persistent (session data lost when application server restarts, or not shared across cluster).
- Session is shared across multiple browsers on same client machine.
- Session is not getting invalidated immediately after specified session timeout interval.
- Unwanted sessions are being created by JavaServer Pages.
- Session data intended for one client is seen by another client.

- A ClassCastException error occurs during failover of a session that contains an Enterprise JavaBeans (EJB) reference.

If your problem is not described here, or none of these steps fixes the problem:
- Review "HTTP session manager troubleshooting tips" on page 151 for general steps on debugging session-manager related problems.
- Review Task overview: Managing HTTP sessions in the *Administering applications and their environment* PDF book for information on how to configure the session manager, and best practices for using it.
- 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.

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

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

    Note: When the servers are on different hosts, ensure that session cookies flow to all the servers by configuring a front-end router such as a Web server with the plug-in or setting the Cookie domain.
- Check the **Cookie path** specified on the SessionManager. Check whether the problem URL is hierarchically below the Cookie path specified. If not correct the Cookie path.
- If the `Cookie maximum age` property is set, ensure that the client (browser) machine's date and time is the same as the server's, including the time zone. If the client and the server time difference is over the ″Cookie maximum age″ then every access would be a new session, since the cookie will ″expire″ after the access.
- If you have multiple Web modules within an enterprise application that track sessions:
  - If you want to have different session settings among Web modules in an enterprise application, ensure that each Web module specifies a different cookie name or path, or
  - If Web modules within an enterprise application use a common cookie name and path, ensure that the HTTP session settings, such as Cookie maximum age, are the same for all Web modules. Otherwise cookie behavior will be unpredictable, and will depend upon which application creates the session. Note that this does not affect session data, which is maintained separately by Web module.
- Check the cookie flow between browser and server:
  1. On the browser, enable ″cookie prompt″. Hit the servlet and make sure cookie is being prompted.
  2. On the server, enable SessionManager trace. Enable tracing for the HTTP session manager component, by using the trace specification ″com.ibm.ws.webcontainer.httpsession.*=all=enabled″. After trace is enabled, exercise your session-using servlet or jsp, then follow the instructions for dumping and browsing the trace output .
  3. Access the session servlet from the browser.
  4. The browser will prompt for the cookie; note the jsessionid.
  5. Reload the servlet, note down the cookie if a new cookie is sent.

6. Check the session trace and look for the session id and trace the request by the thread. Verify that the session is stable across Web requests:
   – Look for **getIHttpsession(...)** which is start of session request.
   – Look for **releaseSesson(..)** which is end of servlet request.

- If you are using URL rewriting instead of cookies:
  – Ensure there are no static HTML pages on your application's navigation path.
  – Ensure that your servlets and JSP files are implementing URL rewriting correctly. For details and an example see the Session tracking options section in the *Developing and deploying applications* PDF book.
- If you are using SSL as your session tracking mechanism:
  – Ensure that you have SSL enabled on your IBM HTTP Server or iPlanet HTTP server.
  – Review the Session tracking options section in the *Developing and deploying applications* PDF book..
- If you are in a clustered (multiple node) environment, ensure that you have session persistence enabled.

HTTP Sessions are not persistent

If your HTTP sessions are not persistent, that is session data is lost when the application server restarts or is not shared across the cluster:

- Check the data source.
- Check the session manager's persistence settings properties:
  – If you intend to take advantage of session persistence, verify that Persistence is set to **Database**.
  – Persistence could also be set to **Memory-to-Memory Replication**.
  – If you are using **Database-based persistence**:
    - Check the JNDI name of the data source specified correctly on SessionManager.
    - Specify correct userid and password for accessing the database.

      Note that these settings have to be checked against the properties of an existing data source in the administrative console. The session manager does not automatically create a session database for you.
    - The data source should be non-JTA, for example, non XA enabled.
    - Check the JVM logs for appropriate database error messages.
    - With DB2, for row sizes other than 4k make sure specified row size matches the DB2 page size. Make sure tablespace name is specified correctly.

Session is shared across multiple browsers on same client machine

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

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

Session is not getting invalidated immediately after specified session timeout interval

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

Unwanted sessions are being created by JavaServer Pages

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

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

Session data intended for one client is seen by another client

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

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

A ClassCastException error occurs during failover of a session that contains an Enterprise JavaBeans (EJB) reference

If you run WebSphere® Application Server for z/OS® Version 6.0.1 and configure a session manager to replicate EJB references, a session failover might trigger display of the following exception in the server region job log:

```
java.lang.ClassCastException: cannot cast class
       org.omg.stub.java.rmi._Remote_Stub to interface javax.ejb.EJBObject
```

The log also displays a null pointer exception. The problem results from the session outbound request, where WebSphere Application Server for z/OS issued a CORBA::COMM_FAILURE exception with a C9C21355 minor code. This behavior occurs because your application server contains all of the following configurations:

1. SAF is both the local operating system, as well as the user registry
2. Attribute propagation is enabled
3. An unauthenticated user initiated the session outbound request

To correct this problem apply the APAR PK06777 fix to WebSphere Application Server for z/OS V6.0.1. You can retain the previously mentioned server configurations.

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

- http://www.ibm.com/software/webservers/appserv/was/support/

HTTP sessions: Resources for learning

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

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

View links to additional information about:

Programming model and decisions
- Improving session persistence performance with DB2
- Persistent client state HTTP cookies specification

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

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

Modifying the default Web container configuration

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

Your application is considered complex if it requires any of the following features:
- Virtual host
- Servlet caching
- Special client request loads
- Persistent HTTP session support
- Special HTTP transport settings

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

1. In the administrative console, click **Servers > Application servers >** *server_name*. Then under Web container settings, click on one of the following:
   a. **Web container**, if your Web application requires a virtual host, other than the default_host, or requires servlet caching.
   b. **Web container transport chains**, if you need to reconfigure your HTTP connections.
   c. **Session management**, if your application requires persistent HTTP session support.
2. If your application handles special client request loads, in the administrative console, click **Servers > Application servers >** *server_name*. Then under Additional Properties, click **Thread Pools** to modify your thread pool settings.
3. If your application requires global settings for internal servlets for WAR files packaged by third-party tools, iIn the administrative console, click **Servers > Application servers >** *server_name* **> Web container settings > Web container**. Then under Additional Properties, click **Custom Properties** and enter the appropriate custom property.

Web container

A Web container handles requests for servlets, JavaServer Pages (JSP) files, and other types of files that include server-side code. The Web container creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks.

The Web server plug-ins, provided by the WebSphere Application Server, help supported Web servers pass servlet requests to Web containers.

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

application server components may not be ready yet. Once the application server is 'ready for e-business', it is completely ready. If application server related calls fail during Servlet.init method, you can either:

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

Web container settings

Use this page to configure the Web container settings.

To view this administrative console page, click **Servers** > **Application servers** > *server_instance* > **Web Container Settings** > **Web container**.

Default virtual host

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

Select a virtual host option:

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

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

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

Enable servlet caching

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

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

Disable servlet request and response pooling

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

Default False

Web container custom properties

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

Global settings for internal servlets

HTTP Transport custom properties can also be set at the Web container level. See HTTP transport custom properties for a description of these properties.

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

Name	Value
fileServingEnabled	true
directoryBrowsingEnabled	true
serveServletsByClassnameEnabled	true

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

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

UTF-8 encoded Uniform Resource Locators (URLs)

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

Name	Value
DecodeUrlAsUTF8	false

Global configuration of servlet listeners

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

Name	Value
listeners	*listener_class*

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

Binary Large Object (BLOB) data type for Oracle databases

The *UseOracleBLOB* custom property creates the HTTP session database table using the Binary Large Object (BLOB) data type for the medium column. This property increases performance of persistent sessions when Oracle databases are used. Due to an Oracle restriction, BLOB support requires use of the

Oracle's oci database driver for more than 4000 bytes of data. You must also ensure that a new sessions table is created before the server is restarted by dropping your old sessions table or by changing the datasource definition to reference a database that does not contain a sessions table. To create a sessions table using the BLOB data type, use the following name-vaule pair:

Name	Value
UseOracleBLOB	true

Detecting session data crossover

The *DebugSessionCrossover* custom property enables code to perform additional checks to verify that only the session associated with the request is accessed or referenced. Messages are logged if any discrepancies are detected. To enable session data crossover detection, use the following name-vaule pair:

Name	Value
DebugSessionCrossover	true

See article, "Problems creating or using HTTP sessions" on page 152, for additional information.

Optimizing Web services client to Web container communication

To improve performance, there is an optimized communication path between a Web services client application and a Web container that are located in the same application server process. Requests from the Web services client that are normally sent to the Web container using a network connection are delivered directly to the Web container using an optimized local path. The local path is available because the Web services client application and the Web container are running in the same process. This optimized communication path is disabled by default. Before enabling this property, make sure that wild cards are not specified for the Web container ports. Use specific ports for the web container when the optimized communication path is enabled. To enable the optimized communication path, use the following name-value pair:

Name	Value
enableInProcessConnections	true

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

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

Name	Value
prependSlashToResource	true

HTTPS requests with an SSL offloader

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

application server checks this indicator to determine if SSL is required. If it determines the request is SSL over HTTP, an HTTPS scheme is chosen.

Name	Value
httpsIndicatorHeader	*Request header key name*

Web module deployment settings

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

To view this administrative console page, click **Applications > Enterprise Application >** *application_instance* **> Manage Modules >** *Web_module_instance*.

URI

Specifies the relative location of the module within the application EAR.

Alternate deployment descriptor

Specifies the alternate deployment descriptor for the module as defined in the application deployment descriptor according to the J2EE specification.

Starting weight

Specifies the order in which modules are started. Lower weighted modules are started before higher weighted modules.

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 product class loaders is Classes loaded with parent class loader first. By specifying Classes loaded with application class loader first, 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 application class loader first. The default is to search in the parent class loader before searching in the application class loader to load a class.

Data type String
Default Classes loaded with parent class loader first

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 panel, click **Applications > Enterprise Applications >** *application_name* **> Context root for Web modules**. This panel is the same as the Context root for Web modules panel 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`.

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 panel, click **Applications > Enterprise Applications >** *application_name* **> Environment entries for Web modules**. This page is the same as the Environment entries for Web modules panel on the application installation and update wizards.

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

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 module `web.xml` file.

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

Description
Specifies information on the environment entry.

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

Web container troubleshooting tips

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

```
WebContainer  A SRVE0161I: IBM WebSphere Application Server - Web Container.
Copyright IBM Corp. 1998-2002
WebContainer  A SRVE0169I: Loading Web Module: [module_name]
ApplicationMg A WSVR0221I: Application started: [application_name]
HttpTransport A SRVE0171I: Transport http is listening on port [port_number]
[server_name] open for e-business in [profile_root]/logs/[server_name]/SystemOut.log
```
- For specific problems that can cause servlets, HTML files, and JavaServer Pages (JSP) files not to be served, see Web resource (JSP file, servlet, HTML file, image) does not display .
- For a detailed trace of the run-time behavior of the Web container, enable trace for the component com.ibm.ws.webcontainer using `com.ibm.ws.webcontainer.*=all:com.ibm.ws.wswebcontainer.*=all`.

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

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

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

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

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 IBM Support page.

Disabling servlet pooling: Best practices and considerations

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

Disabling request and response pooling

- Application is creating threads inside of the application.

  The Servlet 2.4 specification states the following:

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

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

  > **SRV.2.3.3.3 Thread Safety** Implementations of the request and response objects are not guaranteed to be thread safe. This means that they should only be used within the scope of the request handling thread.
  >
  > References to the request and response objects should not be given to objects executing in other threads as the resulting behavior may be nondeterministic. If the thread created by the application uses the container-managed objects, such as the request or response object, those objects must be accessed only within the servlet's service life cycle and such thread itself should have a life cycle within the life cycle of the servlet's service method because accessing those objects after the service method ends may cause undeterministic problems. Be aware that the request and

response objects are not thread safe. If those objects were accessed in the multiple threads, the access should be synchronized or be done through the wrapper to add the thread safety, for instance, synchronizing the call of the methods to access the request attribute, or using a local output stream for the response object within a thread.

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

Configuring session management by level

When you configure session management at the Web container level, all applications and the respective Web modules in the Web container normally inherit that configuration, setting up a basic default configuration for the applications and Web modules below it.

However, you can set up different configurations individually for specific applications and Web modules that vary from the Web container default. These different configurations override the default for these applications and Web modules only.

Note: When you overwrite the default session management settings on the application level, all the Web modules below that application inherit this new setting unless they too are set to overwrite these settings.

1. Open the administrative console.
2. Select the level that this configuration applies to:
   - For the Web container level:
     a. Click **Servers > Application Servers >** *server_instance* **> Web Container Settings > Web Container**.
   - For the enterprise application level:
     a. Click **Applications > Enterprise Applications >** *server_instance*.
   - For the Web module level:
     a. Click **Applications** > **Enterprise Applications >** *server_instance* **> Web Modules**.
     b. Select a Web module from the list of Web modules defined for this application.
3. Under **Additional Properties**, click **Session Management**.
4. Make whatever changes you need to manage sessions.
5. If you are working on the Web module or application level and want these settings to override the inherited Session Management settings, under **General Properties**, select **Overwrite**.
6. Click **Apply** and **Save**.

Configuring session tracking

To configure session tracking, complete the following:

1. Go to the appropriate level of Session Management.
2. Specify which session tracking mechanism you want to pass the session ID between the browser and the servlet:
   - To track sessions with cookies, click **Enable Cookies**.

     To change the cookie settings, click **Modify**.
   - To track sessions with URL rewriting, click **Enable URL Rewriting**.

     If you want to enable protocol switch rewriting, click **Enable protocol switch rewriting.**
   - To track sessions with SSL information, click **Enable SSL ID tracking**.
3. Click **Apply**.
4. Click **Save**.

Serializing access to session data

The Servlet API supports concurrent access to a session in a given server instance. WebSphere Application Server provides an option to prevent the concurrent access to a session in a given server instance so that concurrent modification of a session does not occur in a given server instance. This prevention is achieved by synchronizing the requests based on session. When this feature is turned on, a session is obtained for the request before invoking the servlet and requests are synchronized by locking the session for the servlet execution time. Note that synchronization is based on the memory copy of session. So this feature cannot serialize requests across servers based on session when session affinity fails.

You can also use the serializing access to session data feature to synchronize session objects inside of servlets or JavaServer pages. Applications cannot synchronize session objects inside of their servlets or JavaServer Pages because a deadlock with the session manager may occur. The deadlock occurs because the session manager does not expect the use of more than one locking mechanism. You can ensure that only one request can access the session at a time through the use of the configuration option, Allow serial access.

Use this feature only when concurrent modification of the same session data is possible and is not desirable by the application. This feature has overhead of serializing the requests based on a session.

Do the following to synchronize session access:

1. Select the level of Session Management on which you want to serialize session access.
2. Under Serialize Session access, click **Allow serial access**.
3. In the Maximum wait time box, type the amount of time, in milliseconds, a servlet waits on a session before continuing execution. The default is 120000 milliseconds or two minutes.
4. Select **Allow access on timeout** if you want the servlet to gain access to the session and continue normal execution even if the session is still locked by another servlet. If you do not select this box, the servlet execution will abort when the session request times out.
5. Click **Apply**.
6. Click **Save**.

Session management settings

Use this page to manage HTTP session support. This support includes specifying a session tracking mechanism, setting maximum in-memory session count, controlling overflow, and configuring session timeout.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Web container settings** > **Session management**.

Session tracking mechanism

Specifies a mechanism for HTTP session management.

Mechanism	Function	Default

Enable SSL ID Tracking	Specifies that session tracking uses Secure Sockets Layer (SSL) information as a session ID. Enabling SSL tracking takes precedence over cookie-based session tracking and URL rewriting. **V6.0.x** There are two parameters available if you enable SSL ID tracking: SSLV3Timeout and Secure Authentication Service (SAS). SSLV3Timeout specifies the time interval after which SSL sessions are renegotiated. This is a high setting and modification does not provide any significant impact on performance. The SAS parameter establishes an SSL connection only if it goes out of the Java Virtual Machine (JVM) to another JVM. If all the beans are co-located within the same JVM, the SSL used by SAS does not hinder performance. These are set by editing the `sas.server.properties` and `sas.client.props` files located in the *product_installation_root*\properties directory, where *product_installation_root* is the directory where WebSphere Application Server is installed. **Important:** **V6.0.x** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.	9600 seconds

Enable cookies	Specifies that session tracking uses cookies to carry session IDs. If cookies are enabled, session tracking recognizes session IDs that arrive as cookies and tries to use cookies for sending session IDs. If cookies are not enabled, session tracking uses Uniform Resource Identifier (URL) rewriting instead of cookies (if URL rewriting is enabled). Enabling cookies takes precedence over URL rewriting. Do not disable cookies in the session management facility of the application server that is running the administrative application because this action causes the administrative application not to function after a restart of the server. As an alternative, run the administrative application in a separate process from your applications. Click **Enable cookies** to change these settings.
Enable URL rewriting	Specifies that the session management facility uses rewritten URLs to carry the session IDs. If URL rewriting is enabled, the session management facility recognizes session IDs that arrive in the URL if the encodeURL method is called in the servlet.
Enable protocol switch rewriting	This option is only available when **Enable URL rewriting** is selected. This option specifies that the session ID is added to a URL when the URL requires a switch from HTTP to HTTPS or from HTTPS to HTTP. If rewriting is enabled, the session ID is required to go between HTTP and HTTPS.

Maximum in-memory session count

Specifies the maximum number of sessions to maintain in memory.

The meaning differs depending on whether you are using in-memory or distributed sessions. For in-memory sessions, this value specifies the number of sessions in the base session table. Use the Allow overflow property to specify whether to limit sessions to this number for the entire session management facility or to allow additional sessions to be stored in secondary tables. For distributed sessions, this value specifies the size of the memory cache for sessions. When the session cache has reached its maximum size and a new session is requested, the session management facility removes the least recently used session from the cache to make room for the new one.

Note: Do not set this value to a number less than the maximum thread pool size for your server.

Allow overflow

Specifies that the number of sessions in memory can exceed the value specified by the Max in-memory session count property. This option is valid only in non-distributed sessions mode.

Session timeout
Specifies how long a session can go unused before it is no longer valid. Specify either `Set timeout` or `No timeout`. Specify the value in minutes greater than or equal to two.

The value specified in a Web module deployment descriptor file takes precedence over the administrative console settings. However, the value of this setting is used as a default when the session timeout is not specified in a Web module deployment descriptor. Note that to preserve performance, the invalidation timer is not accurate to the second. When the write frequency is time based, ensure that this value is least twice as large as the write interval.

Security integration
Specifies that when security integration is enabled, the session management facility associates the identity of users with their HTTP sessions

Serialize session access
Specifies that concurrent session access in a given server is not allowed.

Maximum wait time
Specifies the maximum amount of time a servlet request waits on an HTTP session before continuing execution. This parameter is optional and expressed in seconds. The default is 5 seconds. Under normal conditions, a servlet request waiting for access to an HTTP session gets notified by the request that currently owns the given HTTP session when the request finishes.

Allow access on timeout
Specifies whether the servlet is started normally or aborted in the event of a timeout. If this box is checked, the servlet is started normally. If this box is not checked, the servlet execution aborts and error logs are generated.

Cookie settings
Use this page to configure cookie settings for session management.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Web container settings > Session management > Enable cookies**.

Cookie name
Specifies a unique name for the session management cookie. The servlet specification requires the name JSESSIONID. However, for flexibility this value can be configured.

Restrict cookies to HTTPS sessions
Specifies that the session cookies include the secure field. Enabling the feature restricts the exchange of cookies to HTTPS sessions only.

Cookie domain
Specifies the domain field of a session tracking cookie. This value controls whether or not a browser sends a cookie to particular servers. For example, if you specify a particular domain, session cookies are sent to hosts in that domain. The default domain is the server.

Cookie path
Specifies that a cookie is sent to the URL designated in the path. Specify any string representing a path on the server. ″/″ indicates root directory. Specify a value to restrict the paths to which the cookie will be sent. By restricting paths, you prevent the cookie from going to certain URLs on the server. If you specify the root directory, the cookie is sent no matter which path on the given server is accessed.

Cookie maximum age

Specifies the amount of time that the cookie lives on the client browser. Specify that the cookie lives only as long as the current browser session, or to a maximum age. If you choose the maximum age option, specify the age in seconds. This value corresponds to the Time to Live (TTL) value described in the Cookie specification.

Default is the current browser session which is equivalent to setting the value to -1.

Session management custom properties

Custom properties for session management:

CloneSeparatorChange

Use this property to maintain session affinity. The clone ID of the server is appended to session identifier separated by colon. On some Wireless Application Protocol (WAP) devices, a colon is not allowed. Set this property to ″true″ to change clone separator to a plus sign (+).

HttpSessionCloneId

Use this property to change the clone ID of the cluster member. Within a cluster, this name must be unique to maintain session affinity. When set, this name overwrites the default name generated by WebSphere Application Server.

Default clone ID length: 8 or 9

HttpSessionIdLength

Use this property to configure the session identifier length. Do not use an extremely low value; using a low value results in reduced number of combinations possible, thereby increasing risk of guessing the session identifier. In a cluster, all cluster members should be configured with same ID length. Allowed range: 8 to 128. Default length: 23.

HttpSessionReaperPollInterval

Use this property to set a wake-up interval for the process that removes invalid sessions. Default is based on maximum inactive interval set in session management. Allowed value: integer.

NoAdditionalSessionInfo

Set this value to ″true″ to force removal of information that is not needed in session identifiers.

In WebSphere Application Server base edition, a clone ID of -1 is never used; therefore, a clone ID is not included in base edition when this is set. Also, cache ID is not used with nonpersistent sessions; so the cache ID is not included with nonpersistent sessions when this value is set.

SessionIdentifierMaxLength

Use this value to set maximum length that a session identifier can grow. In a cluster, because of fail-over when a request goes to new cluster member, session management appends a new clone ID to the existing clone ID. In a large cluster, if for some reason servers are failing more often, then it is possible that the session identifier length can be more than expected reducing room for URL. So this property helps to find out the condition and take appropriate action to address servers fail-over. When this is specified, message is logged when specified maximum length is reached. Allowed value: integer.

SessionRewriteIdentifier

Use this property to change the key used with URL rewriting. Default key: jsessionid.

Configuring session tracking for Wireless Application Protocol (WAP) devices

Most Wireless Application Protocol (WAP) devices do not support cookies. The preferred way to track sessions for WAP devices is to use URL rewriting. However on most WAP devices, the maximum allowed URL length is 128 characters. With URL rewriting, a session indentifier is added to the URL itself, effectively decreasing the space available for the actual URL and the number of parameters that can be sent on a request.

To reduce the length of session identifier, you can configure key (jsessionid), session ID length and clone ID. To make these configuration changes, complete the following:

1. Open the administrative console.
2. Click **Servers** > **Application Servers>***server_instance***Web Container Settings > Web container**.
3. Under Additional Properties, click **Custom Properties**.
4. Add the appropriate properties from the following list:
   - HttpSessionIdLength
   - SessionRewriteIdentifier
   - HttpSessionCloneId
   - CloneSeparatorChange
   - NoAdditionalSessionInfo
   - SessionIdentifierMaxLength
5. Click **Apply** and **Save**.

Configuring for database session persistence

To configure the session management facility for database session persistence, complete the following:

1. Create and configure a JDBC provider.
2. Create a data source pointing to a database.

   Use the JDBC provider that you defined: **Resources > JDBC Providers >** *JDBC_provider* **> Data Sources > New**. The data source should be non-JTA, for example, non-XA enabled. Note the JNDI name of the data source.

   Point to an existing database.
3. Verify that the correct database is listed under **Resources > JDBC Providers >** *JDBC_provider* **> Data Sources >** *datasource_name*. If necessary, contact your database administrator to verify the correct database name.
4. Go to the appropriate level of Session Management.
5. Under Additional Properties, click **Distributed Environment Settings**
6. Select and click **Database**.
7. Specify the Data Source JNDI name from a previous step.
8. Specify the database user ID and password for accessing the database.
9. Retype the password for confirmation.
10. Configure a table space and page sizes for DB2 session databases.
11. Switch to a multirow schema.
12. Click **OK**.
13. If you want to change the tuning parameters, click **Custom Tuning Parameters** under Additional properties and select a setting or customize a setting.
14. Click **Apply**.
15. Click **Save**.

Switching to a multirow schema

By default, a single session maps to a single row in the database table used to hold sessions. With this setup, there are hard limits to the amount of user-defined, application-specific data that WebSphere Application Server can access.

1. Modify the Session Management facility properties to switch from single to multirow schema.
2. Manually drop the table.

   To drop the table:
   a. Determine which data source configuration Session Management is using.

b. In the data source configuration, look up the database name.
c. Use the database facilities to connect to the database.
d. Drop the SESSIONS table.

Configuring tablespace and page sizes for DB2 session databases

If you are using DB2 for session persistence, you can increase the page size to optimize performance for writing large amounts of data to the database. Page sizes of 8K, 16K, and 32K are supported.

To use a page size other than the default (4K), do the following:

1. If the SESSIONS table already exists, drop it from the DB2 database.
2. Create a new DB2 buffer pool and table space, specifying the same page size (8K, 16K or 32K) for both, and assign the new buffer pool to this table space.

```
DB2 Connect to session
DB2 CREATE BUFFERPOOL sessionBP SIZE 1000 PAGESIZE 8K
DB2 Connect reset
DB2 Connect to session
DB2 CREATE TABLESPACE sessionTS PAGESIZE 8K MANAGED BY SYSTEM
    USING ('D:\DB2\NODE0000\SQL00005\sessionTS.0') BUFFERPOOL sessionBP
DB2 Connect reset
```

   Refer to DB2 product documentation for details.
3. Configure the correct table space name and page size in the Session Management facility. Page size is referred to as *row size* on the Session Management page.)

When the product is restarted, the Session Management facility creates the new SESSIONS table in the specified tablespace based on the indicated page size.

Database settings

Use this page to specify the settings for database session support.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Web container settings > Session management > Distributed environment settings > Database**.

Datasource JNDI name

Specifies the datasource description.

The JNDI name of the non-XA enabled datasource from which session management obtains database connections. For example, if the JNDI name of the datasource is ″jdbc/sessions″, specify ″jdbc/sessions.″ The datasource represents a pool of database connections and a configuration for that pool (such as the pool size). The datasource must already exist as a configured resource in the environment.

User ID

Specifies the user ID for database access.

Password

Specifies the password for database access.

DB2 row size

Specifies the table space page size configured for the sessions table, if using a DB2 database. Possible values are 4, 8, 16, and 32 kilobytes (KB). The default row size is 4KB.

The default row size is 4KB. In DB2, it can be updated to a larger value. This can help database performance in some environments. When this value is other than 4, you must specify table space name to use this property. For 4KB pages, the table space name is optional.

Table space name
Specifies that table space to be used for the sessions table.

This value is required when the DB2 page size is other than 4KB.

Use multi row schema
Specifies that each instance of application data be placed in a separate row in the database, allowing larger amounts of data to be stored for each session. This action can yield better performance in certain usage scenarios. If using multi row schema is not enabled, instances of application data can be placed in the same row.

Multirow schema considerations
WebSphere Application Server supports the use of a multirow schema option in which each piece of application specific data is stored in a separate row of the database. With this setup, the total amount of data you can place in a session is now bound only by the database capacities. The only practical limit that remains is the size of the session attribute object.

The multirow schema potentially has performance benefits in certain usage scenarios, such as when larger amounts of data are stored in the session but only small amounts are specifically accessed during a given servlet processing of an HTTP request. In such a scenario, avoiding unneeded Java object serialization is beneficial to performance.

Understand that switching between multirow and single row is not a trivial proposition.

In addition to allowing larger session records, using multirow schema can yield performance benefits. However, it requires a little work to switch from single-row to multirow schema, as shown in the instructions below.

Coding considerations and test environment

Consider configuring direct single-row usage to one database and multirow usage to another database while you verify which option suits your application needs. (Do this in code by switching the data source used; then monitor performance.)

Programming issue	**Application scenario**
Reasons to use single-row	• You can read or write all values with just one record read and write. • This takes up less space in a database because you are guaranteed that each session is only one record long.
Reasons **not** to use single-row	2-megabyte limit of stored data per session.
Reasons to use multirow	• The application can store an unlimited amount of data; that is, you are limited only by the size of the database and a 2-megabyte-per-record limit. • The application can read individual fields instead of the whole record. When large amounts of data are stored in the session but only small amounts are specifically accessed during servlet processing of an HTTP request, multirow sessions can improve performance by avoiding unneeded Java object serialization.
Reasons **not** to use multirow	If data is small in size, you probably do not want the extra overhead of multiple row reads when you can store everything in one row.

In the case of multirow usage, design your application data objects not to have references to each other, to prevent circular references. For example, suppose you are storing two objects A and B in the session using HttpSession.put(..) method, and A contains a reference to B. In the multirow case, because objects are stored in different rows of the database, when objects A and B are retrieved later, the object graph between A and B is different than stored. A and B behave as independent objects.

Configuring Web module class loaders

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

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

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` on the settings page of an application class loader, then the application class loader loads the Web module contents as well as the enterprise bean (EJB) modules, shared libraries, resource adapter archives (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.

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

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

   Settings such as **Reload classes when application files are updated**, **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	Causes the class loader to search in the parent application class loader first to load a class. This is the standard for Development Kit class loaders and WebSphere Application Server class loaders. **Tip:** If classes and resources needed by the Web module cannot be accessed by the application class loader, but can be accessed by the WAR class loader, specify `Classes loaded with application class loader first`. If the application class loader cannot find a class, the class loader delegates the request to find the class to its parent, the WebSphere Application Server extensions class loader. If the WebSphere Application Server extensions class loader cannot find the class, the class loader delegates the request to its parent, the bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine. Requests can only go to a parent class loader; they cannot go to a child class loader. Thus, if `Classes loaded with parent class loader first` is specified, the WAR class loader never receives a request to load a class.
Classes loaded with application class loader first	Causes the class loader to search in the WAR class loader first to load a class. By specifying `Classes loaded with application class loader first`, your WAR class loader can override classes contained in the parent application class loader. **Attention:** Specifying the `Classes loaded with application class loader first` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

3. Click **OK**.

Save the changes to the administrative configuration.

Chapter 8. Portlet applications

Task overview: Managing portlets

You can use this task to manage deployed portlet applications.

Before you begin this task, you must have a portlet application installed. See “Installing application files” on page 28 for additional information.

You can complete the following steps to manage portlets.

- Render a portlet.
  - Access a single portlet using “Portlet Uniform Resource Locator (URL) addressability” on page 182.
  - Access multiple portlets using “Portlet aggregation using JavaServer Pages” on page 176.
- Change the location of “Portlet preferences” on page 183. By default, portlet preferences for each portlet window are stored in a cookie. However, you can change the location of where to store portlet preferences.
- Disable URL addressability. By default, you can access a portlet through an Uniform Resource Locator (URL), however, you can disable this feature.
- Enable portlet fragment caching. Portlet fragment caching is disabled by default.

Portlets

Portlets are reusable Web modules that provide access to Web-based content, applications, and other resources. Portlets can run on WebSphere Application Server because it has an embedded JSR168 Portlet Container. You can assemble portlets into a larger portal page, with multiple instances of the same portlet displaying different data for each user.

From a user’s perspective, a portlet is a window on a portal site that provides a specific service or information, for example, a calendar or news feed. From an application development perspective, portlets are pluggable Web modules that are designed to run inside a portlet container of any portal framework. You can either create your own portlets or select portlets from a catalog of third-party portlets.

Each portlet on the page is responsible for providing its output in the form of markup fragments to be integrated into the portal page. The portal is responsible for providing the markup surrounding each portlet. In HTML, for example, the portal can provide markup that gives each portlet a title bar with minimize, maximize, help, and edit icons.

You can also include portlets as fragments into servlets or JavaServer Pages files. This provides better communication between portlets and the J2EE Web technologies provided by the application server.

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

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

Also if you use RAD version 6 to create portlets, note that portlets created by using the Struts Portlet Framework are not supported on WebSphere Application Server.

Portlet applications

If the portlet application is a valid Web application written to the Java Portlet API, the portlet application can operate on both the Portal Server and the WebSphere Application Server without requiring any

changes. A JSR 168 compliant portlet application must not use extended services provided by WebSphere Portal to operate on the WebSphere Application Server.

Portlet container

The *portlet container* is the runtime environment for portlets using the JSR 168 Portlet Specification, in which portlets are instantiated, used, and finally destroyed. The JSR 168 Portlet API provides standard interfaces for portlets. Portlets based on this JSR 168 Portlet Specification are referred to as standard portlets.

A simple portal framework is provided by the PortletServingServlet servlet. The PortletServingServlet servlet registers itself for each Web application that contains portlets. You can use the PortletServingServlet servlet to directly render a portlet into a full browser page by a URL request and invoke each portlet by its context root and name. See “Portlet Uniform Resource Locator (URL) addressability” on page 182 for additional information. If you want to aggregate multiple portlets on the page, you need to use the aggregation tag library. See the article “Portlet aggregation using JavaServer Pages” for additional information. The PortletServingServlet servlet can be disabled in an extended portlet deployment descriptor called the ibm-portlet-ext.xmi file.

Remote request dispatcher support for portlets

The remote request dispatcher (RRD) support allows the invocation of portlets outside of the current Java virtual machine (JVM) within an Network Deployment single core group environment. The request related data is passed to the remote JVM where the portlet is invoked. The response is transmitted back and processed on the local JVM. Thus it guarantees that URLs contained in the portlet markup are created according to the local portal context.

Portlet container settings

Use this page to configure and manage the portlet container of this application server.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Portlet Container Settings** > **Portlet container**.

Enable portlet fragment cache

Specifies whether to create a cached entry when a portlet is invoked, similar to servlet caching of the Web container settings.

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

Portlet aggregation using JavaServer Pages

The aggregation tag library generates a portlet aggregation framework to address one or more portlets on one page If you write JavaServer Pages, you can aggregate multiple portlets on one page using the aggregation tag library. This tag library does not provide full featured portal aggregation implementation, but provides a good migration scenario if you already have aggregating servlets and JavaServer Pages and want to switch to portlets.

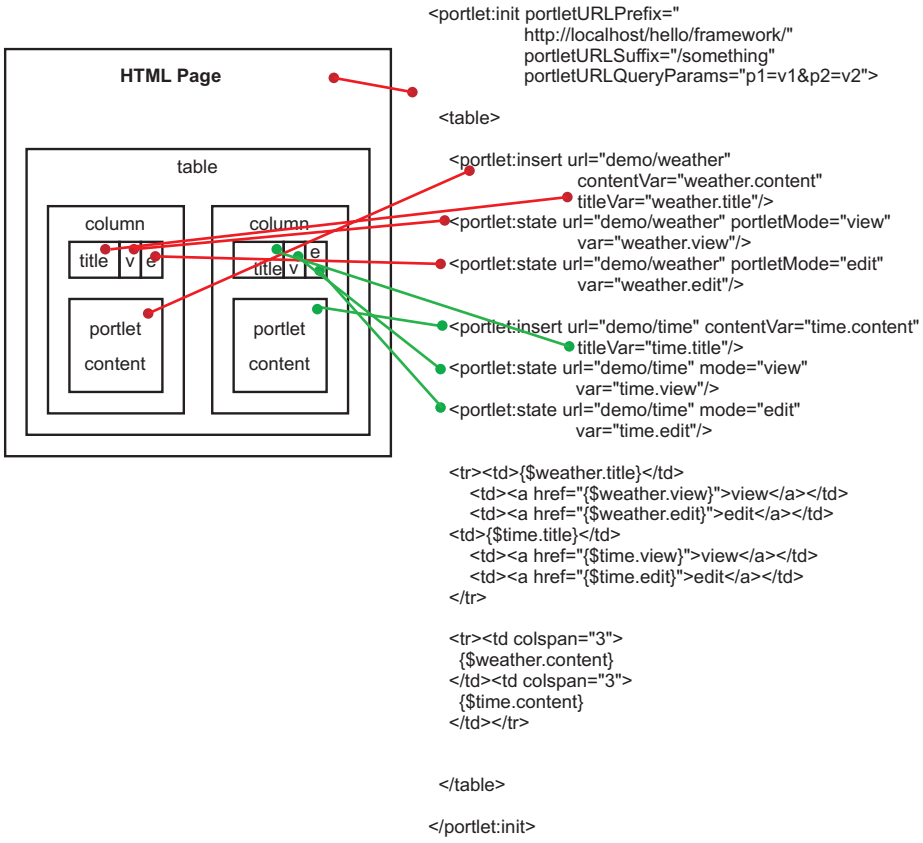
To allow the customer to create a simple portal aggregation, the aggregation tag library also provides the following features.

- Invoke a portlet’s action method
- Render multiple portlets on one page
- Provide links to change the portlet’s mode or window state
- Display the portlet’s title

- Retain the portlet cookie state

The aggregation tag library and JavaServer Pages that use the aggregation tag library will only work with the WebSphere Application Server portlet container implementation because the protocol between the tags and the container is not standardized.

The following diagram depicts how an HTML page would look like and what tags are used in order to create the page. See “Aggregation tag library attributes” for information on the aggregation tag library attributes.



When you use the aggregation tag library, you must set the portletUrlPrefix attribute of the init tag to the aggregating application. You need to:

- Ensure that the portletUrlPrefix attribute is set to the following in the aggregator page.

  `"http://" + <server_address> + ":" + <server_port> + "/" + <aggregator context> + "/" <aggregator mapping>`
- Reference the aggregation JSP page within the web.xml file through a servlet mapping ending with `/*`. For example, `/aggregation/*`

When aggregating multiple portlets on a single page, special care must be used with the naming conventions of form attribute names in your portlets. Because your portlets are all on the same page, they all share the same HttpServletRequest. When one portlet is viewed the entire page is refreshed and form data is re-posted. Therefore, if there are multiple portlets that are aggregated on a single page with the same form attribute names, there could be logic corruption when form data is re-posted.

Aggregation tag library attributes

The aggregation tag library is used to aggregate multiple portlets on one page. This topic describes the attributes within the aggregation tag library.

Supported arguments include:

init

This tag initializes the portlet framework and has to be used in the beginning of the JSP. All other tags described in this section are only valid in the body of this tag, therefore the init tag usually encloses the whole body of a JSP. In case the current URL contains an action flag the action method of the corresponding portlet is called. The state and insert tags are sub-tags of the init tag.

The init tag has the following attributes:

- portletURLPrefix="<any string>"

  This URL defines the prefix used for PortletURLs. Portlet URLs are created either by the state tag or within a portlet's render method, which is called by using the insert tag. This is a required attribute.
- portletURLSuffix="<any string>"

  This URL defines the suffix used for PortletURLs. Portlet URLs are created either by the state tag or within a portlet's render method, which is called by using the insert tag. This is attribute optional.
- portletURLQueryParams="<any string>"

  This URL defines the query parameters used for PortletURLs. Portlet URLs are created either by the state tag or within a portlet's render method, which is called by using the insert tag. This is attribute optional.

state

The state tag creates a URL pointing to the given portlet using the given state. You can place this URL either into a variable specified by the var attribute or you can write it directly to the output stream. This tag is useful to create URLs for HTML buttons, images, and other items such that when the URL is invoked, the state changes defined in the URL are applied to the given portlet.

The state tag has the following attributes:

- url="<context>/<portlet-name>"

  Identifies the portlet for this tag by using the context and portlet-name to address the portlet. This attribute is required.
- windowId="<any string>"

  Defines the window ID for the portlet URL created by this tag. This is attribute optional.
- var="<any string>"

  If defined the URL is written into a variable with the given scope and name, not to the output stream. This is attribute optional.
- scope="page|request|session|application"

  This attribute is only valid if the var attribute is specified. If defined, the URL is not written to the output stream but a variable is created in the given scope with the given name. The default is page. This is attribute optional.
- portletMode="view|help|edit|<custom>"

  This attribute sets the portlet mode.
- portletWindowState="maximized|minimized|normal|<custom>"

  This attribute sets the window state.
- action="true/false"

  This attribute defines whether this is an action URL. This is attribute optional. The default is false.

urlParam

Adds a render parameter to the newly created URL.

The urlParam tag has the following attributes:

- name="<any string>"

  Indicates the name of the parameter. This is attribute required.

- value=”<any string>”

  Indicates the value of the parameter. This is attribute required.

insert

This tag calls the render method of the portlet and retrieves the content as well as the title. You can optionally place the content and title of the specified portlet into variables using the contentVar and titleVar attributes.

The insert tag has the following attributes:

- url=”<context>/<portlet-name>” (mandatory) Identifies the portlet for this tag by using the context and portlet-name to address the portlet

  This is attribute required.
- windowId=”<any string>”

  Defines the window ID of the portlet. This is attribute optional.
- contentVar=”<any string>”

  If defined, the portlet’s content is not written to the output stream but written into a variable with the given scope and name. This is attribute optional.
- contentScope=”page|request|session|application”

  This attribute is only valid if the contentVar tag is used. If defined, the portlet’s content is written into a variable with the given scope and name, not to the output stream. The default is page. This is attribute optional.
- titleVar=”<any string>”

  If defined the portlet’s title is written into a variable with the given scope and name. If it is not defined, the title is ignored and not written to the output stream. This is attribute optional.
- titleScope=”page|request|session|application”

  This attribute is only valid if titleVar tag is used. If defined, the portlet’s title is written into a variable with the given scope and name, not to the output stream. The default is page. This is attribute optional.

Example: Using the portlet aggregation tag library

You can use the aggregation tag library to aggregate multiple portlets to have multiple and different content on one page. The library can be used by every JavaServer Pages (JSP) file that has been included by a servlet.

To use the portlet aggregation tag library, you must declare the tag-lib at the top of the JSP file using, `<%@ taglib uri="http://ibm.com/portlet/aggregation" prefix="portlet" %>`, as in the following example. The following JSP file example shows how to aggregate portlets on one page.

```
<%@ taglib uri="http://ibm.com/portlet/aggregation" prefix="portlet" %>
<%@ page isELIgnored="false" import="java.util.Enumeration"%>

<portlet:init portletURLPrefix="/dummy/portletTagTest/" portletURLSuffix="/more" portletURLQueryParams="p1=v1&p2=v2">


   <portlet:insert url="worldclock/StdWorldClock" contentVar="worldclockcontent" titleVar="worldclocktitle"/>
    <portlet:state url="worldclock/StdWorldClock" portletMode="view" var="worldclockview"
       portletWindowState="maximized">
        <portlet:urlParam name="namea" value="valuea"/>
        <portlet:urlParam name="nameb" value="valueb"/>
    </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" portletMode="edit" var="worldclockedit" portletWindowState="normal">
        <portlet:urlParam name="name1" value="value1"/>
        <portlet:urlParam name="name2" value="value2"/>
    </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" portletMode="view" var="worldclockmin"
       portletWindowState="minimized">
        <portlet:urlParam name="namemin" value="valuemin"/>
        <portlet:urlParam name="namemin" value="valuemin"/>
```

```
    </portlet:state>

    <portlet:insert url="worldclock/StdWorldClock" windowId="min" contentVar="simplecontent" titleVar="simpletitle"/>
    <portlet:state url="worldclock/StdWorldClock" windowId="min" portletMode="view" var="simpleview"
  portletWindowState="maximized">
        <portlet:urlParam name="name3" value="value3"/>
        <portlet:urlParam name="name4" value="value4"/>
        <portlet:urlParam name="name5" value="value5"/>
        <portlet:urlParam name="name5" value="value5a"/>
        <portlet:urlParam name="name5" value="value5b"/>
        <portlet:urlParam name="name5" value="value5c"/>
    </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" windowId="min" portletMode="edit" var="simpleedit"
  action="true" portletWindowState="normal">
        <portlet:urlParam name="name6" value="value6"/>
        <portlet:urlParam name="name6" value="value6z"/>
            </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" windowId="min" portletMode="view" var="simplemin"
  portletWindowState="minimized">
        <portlet:urlParam name="name1" value="value1"/>
        <portlet:urlParam name="name2" value="value2"/>
    </portlet:state>

    <portlet:insert url="test/TestPortlet1" contentVar="testcontent" titleVar="testtitle"/>
    <portlet:state url="test/TestPortlet1" portletMode="view" var="testview" portletWindowState="maximized"/>
    <portlet:state url="test/TestPortlet1" portletMode="edit" var="testedit" portletWindowState="maximized"/>


<!--  This table is the outtermost table for creating two-column portal layout  -->
<TABLE border="0" CELLPADDING="3" CELLSPACING="8" WIDTH="100%">
<TR>
<TD VALIGN="top">

<!--  This table is the top portlet in the first column -->

  <table border="0" width="100%" CELLPADDING="3" CELLSPACING="0" CLASS="portletTable" SUMMARY="portlet top left">
    <tr><td class="portletTitle" NOWRAP>worldclock title:${worldclocktitle}</td>
        <td CLASS="portletTitleControls"  NOWRAP>
            <a href="${worldclockview}">view</a>
            <a href="${worldclockedit}">edit</a>
            <a href="${worldclockmin}">minimize</a>
        </td>
    </tr>
    <tr>
    <td CLASS="portletBody" COLSPAN="2">
      ${worldclockcontent}
     </td>
    </tr>
  </table>

  <BR/>

<!--  This table is the bottom portlet in the first column -->

  <table border="0" width="100%" CELLPADDING="3" CELLSPACING="0" CLASS="portletTable" SUMMARY="portlet bottom left">
    <tr>
    <td class="portletTitle" NOWRAP>test title:${testtitle}</td>
        <td CLASS="portletTitleControls" NOWRAP>
            <a href="${testview}">view</a>
            <a href="${testedit}">edit</a>
        </td>
    </tr>
    <tr>
    <td CLASS="portletBody" COLSPAN="2">
      ${testcontent}
     </td>
    </tr>
```

```
  </table>


  </TD>



  <TD VALIGN="top">


<!--  This table is the top portlet in the second column -->

  <table border="0" width="100%" CELLPADDING="3" CELLSPACING="0" CLASS="portletTable" SUMMARY="portlet top right">
  <tr>
        <td class="portletTitle" NOWRAP>simple title:${simpletitle}</td>
        <td CLASS="portletTitleControls" NOWRAP>
            <a href="${simpleview}">view</a>
            <a href="${simpleedit}">edit</a>
            <a href="${simplemin}">minimize</a>
        </td>
    </tr>
    <tr>
    <td CLASS="portletBody" COLSPAN="2">
      ${simplecontent}
     </td>
    </tr>
  </table>

    </TD>
    </TR>
    </table>


</portlet:init>
```

You can include the following formatting to the previous example JSP file immediately after declaring the tag library.

```
<STYLE TYPE="TEXT/CSS">
BODY {
    font-family:Verdana,sans-serif; font-size:70%
}
.portletTitle {
    text-align: left;border-top: #000000 1px solid; border-bottom: #000000 1px solid; FONT-SIZE: 60.0%;
    COLOR: #ffffff; FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif; BACKGROUND-COLOR: #5495d5;
}
.portletTitleControls {
    text-align: right;border-top: #000000 1px solid; border-right: #000000 1px solid; border-bottom: #000000
    1px solid; FONT-SIZE: 60.0%; COLOR: #ffffff; FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif;
    BACKGROUND-COLOR: #5495d5;
}
.portletTitleControls A {
    COLOR: #ffffff; text-decoration:none; border:#5495d5 1px solid;border-left:white 1px solid;
    padding-left:0.5em; padding-right:0.5em;
}
.portletTitleControls A:hover {
    COLOR: #ffffff; text-decoration:none; border-top:white 1px solid;
   border-bottom:white 1px solid;border-right:white 1px solid;
}
.minimizeControl {
    font-weight:bold; font-size:100%;
}
.portletTable {
    border-left: gray 1px solid;
    border-bottom: gray 1px solid;
    border-right: gray 1px solid;
}
```

```
.portletBody {
    font-family:Verdana,sans-serif; font-size:70%
}

</STYLE>
```

Portlet Uniform Resource Locator (URL) addressability

You can request a portlet directly through a Uniform Resource Locator (URL) to display its content without portal aggregation. The PortletServingServlet servlet registers each Web application that contains portlets. It is similar to the FileServingServlet servlet of the Web container that serves resources. The PortletServingServlet servlet allows you to directly render a portlet into a full browser page by a URL request.

You can invoke each portlet by its context root and name with the URL mapping `/<portlet-name>` that is created for each portlet. The context root and name has the following format:

```
http://<host>:<port>/<context-root>/<portlet-name>
For example, http://localhost:9080/portlets/TestPortlet1
```

The context root identifies the Web archive (WAR) file that contains the portlet. The portlet name uniquely identifies the portlet with a portlet application of a WAR file. The DefaultDocumentFilter servlet only supports HTML, UTF8 encoding and an extended URL form based on the basic URL form as shown above.

You can only display one portlet at a time using the PortletServingServlet servlet. If you want to aggregate multiple portlets on the page, you need to use the aggregation tag library. See the article “Portlet aggregation using JavaServer Pages” on page 176 for additional information.

Because a portlet only delivers fragment output whereas a servlet usually delivers document output, a mechanism is introduced to convert the fragment into a document, called the PortletDocumentFilter filter. By default, the PortletDocumentFilter filter only supports converting HTML. The conversion is implemented using a servlet filter before the PortletServingServlet servlet is initiated to return the portlet‘s content inside of a document. This default document servlet filter only applies to URL requests, not for includes or forwards using the RequestDispatcher method. You can create servlet filters to support other markups additional document servlet filters. See the article, “Converting portlet fragments to an HTML document” on page 185, for additional information.

The PortletServingServlet servlet does not persist portlet preferences in a XML file or database. It places the portlet preferences directly into a cookie to store the preferences persistently. See the article, “Portlet preferences” on page 183, for additional information on how to change this behavior.

Portlet URL syntax

You can add additional portal context such as portlet mode or window state. You can access the PortletServingServlet servlet by using a URL mapping that has the following structure:

```
http://host:port/context/portlet-name [/portletwindow[/ver [/action] [/mode] [/state] [rparam][/?name]]]
```

Any differing URL structure results in a com.ibm.wsspi.portletcontainer.InvalidURLException exception. Empty strings are not permitted as parameter values and creates an InvalidURLException exception. The following is a list of valid parameters:

http:// host:port/context/*portlet-name*
: This is the minimum URL required to access a portlet. A default portlet window called ‘default’ is created. The *portlet-name* variable is case-sensitive.

/*portletwindow*
: This parameter identifies the portlet window. You must set this parameter if you choose to add more portal context information to the URL.

/ver=*major.minor*

This optional parameter is used to define the version of the portlet API that is used. You must set this parameter if you choose to add more portal context information to the URL. Only the version '1.0' is allowed. Any differing version creates an InvalidURLException exception.

/action

This is a required parameter if you call the action method of the portlet. The action parameter causes the action process of the portlet to be called. After the action has been completed, a redirect is automatically issued to call the render process. To control the subsequent render process, a document servlet filter can set a request attribute with name 'com.ibm.websphere.portlet.action' and value 'redirect' to specify that the portlet serving servlet directly returns after action without calling the render process.

/mode=view | edit | help | *custom-mode*

This optional parameter defines the portlet mode that is used to render the portlet. The default mode is 'view'. The value is not case-sensitive, For example, 'View', 'view' or 'VIEW' results in the same mode.

/state=normal | maximized | minimized | *custom-state*

This optional parameter defines the window state that is used to render the portlet. The default state is 'normal'. The value is not case-sensitive, For example, 'Normal', 'normal', or 'NORMAL' results in the same state.

*** [/rparam=***name* ***[=***value***]]**

This optional parameter specifies render parameters for the portlet. Repeat this parameter chain to provide more than one render parameter. For example, /rparam=invitation/ rparam=days=Monday=Tuesday

?name=*value***&***name2***=***value2* **...**

Query parameters may follow optionally. They are not explicitly supported by the portlet container, but they do not invalidate the URL format.

The following list includes examples of valid URLs:

- http:// localhost:9080/sample/WorldClock
- http:// localhost:9080/sample/WorldClock/myPortlet/ver=1.0/mode=edit/rparam=timezone=UTC
- http:// localhost:9080/sample/WorldClock/myPortlet/ver=1.0/action/state=maximized?timezone=UTC

Portlet preferences

Preferences are set by portlets to store customized information. By default, the PortletServingServlet servlet stores the portlet preferences for each portlet window in a cookie. However, you can change the location to store them in either a session, an .xml file, or a database.

Storing portlet preferences in cookies

The attributes of the cookie are defined as follows:

Path

context/*portlet-name*/*portletwindow*

Name:

The name of the cookie has the fixed value of **PortletPreferenceCookie**.

Value

The value of the cookie contains a list of preferences by mapping to the following structure:

```
*['/' pref-name *['=' pref-value]]
```

All preferences start with '/' followed by the name of the preference. If the preference has one or more values, the values follow the name separated by the '=' character. A null value is represented by the string '#*!0_NULL_0!*#'. As an example, the cookie value may look like, /locations=raleigh=boeblingen/regions=nc=bw

Customizing the portlet preferences storage

You can override how the cookie is handled to store preferences in a session, an .xml file or database. To customize the storage, you must create a filter, servlet or JavaServer Pages file as new entry point that wraps the request and response before calling the portlet. Examine the following example wrappers to understand how to change the behavior of the PortletServingServlet to store the preferences in a session instead of cookies.

The following is an example of how the main servlet manages the portlet invocation.

```
public class DispatchServlet extends HttpServlet
{
    ...
    public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
 response.setContentType("text/html");

 // create wrappers to change preference storage
 RequestProxy req = new RequestProxy(request);
 ResponseProxy resp = new ResponseProxy(request, response);

 // create url prefix to always return to this servlet
                ...
                req.setAttribute("com.ibm.wsspi.portlet.url.prefix", urlPrefix);

 // prepare portlet url
 String portletPath = request.getPathInfo();
                ...

 // include portlet using wrappers
 RequestDispatcher rd = getServletContext().getRequestDispatcher(modifiedPortletPath);
                rd.include(req, resp);
    }
}
```

In the following example, the request wrapper changes the cookie handling to retrieve the preferences out of the session.

```
public class RequestWrapper extends HttpServletRequestWrapper
{
    ...
    public Cookie[] getCookies() {
        Cookie[] cookies = (Cookie[]) session.getAttribute("SessionPreferences");
        return cookies;
    }
}
```

In the following example, the response wrapper changes the cookie handling to store the preferences in the session:

```
public class ResponseProxy extends HttpServletResponseWrapper
{
   ...
    public void addCookie(Cookie cookie) {
        Cookie[] oldCookies = (Cookie[]) session.getAttribute("SessionPreferences");
        int newPos = (oldCookies == null) ? 0 : oldCookies.length;
        Cookie[] newCookies = new Cookie[newPos+1];
        session.setAttribute("SessionPreferences", newCookies);

        if (oldCookies != null) {
```

```
        System.arraycopy(oldCookies, 0, newCookies, 0, oldCookies.length);
        }
        newCookies[newPos] = cookie;
    }
}
```

Portlet deployment descriptor extensions

Extensions for the portlet deployment descriptor are defined within a file called ibm-portlet-ext.xmi. This deployment descriptor is an optional descriptor that you can use to configure WebSphere extensions for the portlet application and its portlets. For example, you can disable the PortletServingServlet servlet for the portlet application in the extended portlet deployment descriptor.

The ibm-portlet-ext.xmi extension file is loaded during application startup. If there are no extension files specified with this setting, the portlet container's default values are used.

The default for the portletServingEnabled attribute is true. The following is an example of how to configure that a PortletServingServlet servlet is not created for any portlet on the portlet application.

```
<?xml version="1.0" encoding="UTF-8"?>
<portletappext:PortletApplicationExtension xmi:version="1.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:portletappext="portletapplicationext.xmi"
    xmlns:portletapplication="portletapplication.xmi"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmi:id="PortletApp_ID_Ext"
    portletServingEnabled="false">
  <portletappext:portletApplication href="WEB-INF/portlet.xml#myPortletApp"/>
</portletappext:PortletApplicationExtension>
```

Converting portlet fragments to an HTML document

A portlet only delivers fragment output whereas a servlet typically delivers document output. However, you can use the PortletServingServlet servlet, which is similar to the FileServingServlet servlet, to address portlets like servlets. A default document servlet filter, the DefaultFilter filter, is applied to the PortletServingServlet servlet to return the portlet's content inside of a document. This filter only applies to requests, not to includes or forwards using the RequestDispatcher method. A servlet filter that is used to embed the portlet's content into a document is called the document servlet filter. You can define additional document servlet filters in a .xml file.

The FilterRequestHelper attribute within com.ibm.wsspi.portletcontainer.util is provided to assist the document servlet filters in analyzing a request regarding filter chain and portlet information. It is used in supporting dynamic portlet titles, as a marker for redirection for document servlet filters and to ensure that document conversion is completed once.

Adding a new document servlet filter

The filter capability is a server feature, therefore all filters must be installed into the server to use the filter capability of the server. The filters need to be available in any classes or library directory on a server level. You must also register the filter in a plugin.xml file within the root of a Java archive (JAR) file. The following is an example of how to register the filter in a plugin.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin id="sample.plugin" name="Customer_Plugin" provider-name="Customer" version="1.0.0">
  <extension point="com.ibm.ws.portletcontainer.portlet-document-filter-config">
    <portlet-document-filter class-name="sample.filter.CustomFilter" order="200" />
  </extension>
</plugin>
```

Dynamic portlet titles

The PortletServingServlet servlet supports dynamic portlet titles by providing the dynamic title as a request attribute, FilterRequestHelper.DYNAMIC_TITLE. This attribute returns the dynamic portlet title if it has been set by the portlet, otherwise it returns the static portlet title of the portlet.xml file if defined.

The FilterRequestHelper is used to assist in controling of the dynamic portlet title. The following constant is defined, `DYNAMIC_TITLE = 'javax.portlet.title'`

The DefaultFilter uses this request attribute to set the document title while converting the fragment into a document. If the filter wants to support browser caching or dynamic portlet titles, the complete portlet content must be cached

Redirection for document servlet filters

A document servlet filter can set a marker as request attribute, FilterRequestHelper.REDIRECT. This marker ensures that the portlet container returns to the document servlet filter after the portlet action has been called prior to any render calls. The following constants are defined, `REDIRECT = 'com.ibm.websphere.portlet.action'` and `REDIRECT_VALUE = 'redirect'`. The DefaultFilter uses this request attribute to provide special cache handling for the portlet rendering call to support dynamic title.

Document conversion

The conversion of the portlet's fragment into a valid document must be completed only once. Therefore each document servlet filter must ensure that the fragment has not yet been converted to a document previously. If the document servlet filter converts the fragment to a document, the request attribute FilterRequestHelper.DOCUMENT must be set to FilterRequestHelper.DOCUMENT_VALUE. This request attribute marks whether the conversion still needs to be completed. The following constants are defined, `DOCUMENT = 'com.ibm.websphere.portlet.filter'` and `DOCUMENT_VALUE = 'document'`. The DefaultFilter uses this request attribute to check whether it should convert the fragment to an Hypertext Markup Language (HTML) document. For example, this allows another document servlet filter in front to convert the fragment into a valid Wireless Markup Language (WML) document instead.

Portlet and PortletApplication MBeans

The MBeans of type portlet and portletapplication provide information about a given portlet application and its portlets. Through the MBean of type portletapplication, you can retrieve a list of names of all portlets that belong to a portlet application. By querying the MBean of type portlet with a given portlet name, you can retrieve portlet specific information from the MBean of type portlet.

Each MBean that corresponds to a portlet or portlet application is uniquely identifiable by its name. Portlet applications are not required to have a name set within the portlet.xml. Thus for MBeans of type portletapplication, the Web module name concatenated with the string ″_portletapplication″ has been chosen as the MBean name. The name chosen for the MBean of type portlet is the name of the MBean of type portletapplication the portlet belongs to, concatenated with the porlet name. A full stop separates the preceding Web module name from the portlet name. For more information about the Portlet and PortletApplication MBean types in the Generated API documentation. The generated API documentation is available in the information center table of contents from the path, **Reference** > **Administrator** > **API documentation** > **MBean interfaces**.

The following code is an example of how to invoke the MBean of type portletapplication for an application with the name ″Bookmark″.

```
String myPortletApplicationName = "Bookmark_war_portletapplication";
This name is composed by the Web module name concatenated with the substring "_portletapplication"


com.ibm.websphere.management.AdminService adminService =
```

```
  com.ibm.websphere.management.AdminServiceFactory.getAdminService();
javax.management.ObjectName on =
  new ObjectName("WebSphere:type=PortletApplication,name=" + myPortletApplicationName + ",*");

Iterator onIter = adminService.queryNames(on, null).iterator();
while(onIter.hasNext())
{
 on = (ObjectName)onIter.next();
}

String ctxRoot = (java.lang.String)adminService.getAttribute(on, "webApplicationContextRoot");
```

In the previous example, the MBeanServer is first queried for an MBean of type portletapplication. If this query is successful, the attribute webApplicationContextRoot is retrieved on that MBean or the first MBean that is found and the result is stored in the variable ctxRoot. This variable now contains the context root of the Web application that contains the portlet application that was searched. For example, this may be something like, ″/bookmark″.

The next code example demonstrates how to invoke the MBean of type portlet for a portlet with the name ″BookmarkPortlet″.

```
String myPortletName = "Bookmark_war_portletapplication.BookmarkPortlet";
This name is composed by the name of the MBean of type portletapplication and
the portlet name, separated by a full stop because the same portlet name may
be used within different Web modules, but must be unique within the system.


com.ibm.websphere.management.AdminService adminService =
  com.ibm.websphere.management.AdminServiceFactory.getAdminService();
javax.management.ObjectName on =
  new ObjectName("WebSphere:type=Portlet,name=" + myPortletName + ",*");
Iterator iter = adminService.queryNames(on, null).iterator();

while(iter.hasNext())
{
 on = (ObjectName)iter.next;
}

java.util.Locale locale = (java.util.Locale) adminService.getAttribute(on, "defaultLocale");
```

The locale returned by the method getAttribute method for the MBean is the default locale defined for this portlet.

Chapter 9. SIP applications

Providing real time collaboration with SIP applications

Follow these procedures for creating SIP applications and configuring the SIP container.

Session Initiation Protocol (SIP) is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging. A *SIP application* is a Java program that uses at least one Session Initiation Protocol (SIP) servlet. A SIP servlet is a Java-based application component that is managed by a SIP servlet container.

The servlet container is a part of an application server that provides the network services over which requests and responses are received and sent. The servlet container decides which applications to invoke and in what order. A servlet container also contains and manages a servlet through its lifecycle.

This topic is divided into the following subsections:

- Configure the SIP container: Information and instructions for configuring SIP container properties and timers.
- Developing SIP applications: Reference information for developers.
- Deploying SIP applications: Information for installing, starting, and stopping, your applications.
- Securing SIP applications: Instructions for enabling security providers and setting up a trust association interceptor (TAI).
- Tracing a SIP container: Troubleshoot SIP applications through traces on the SIP container.

SIP applications

A *SIP application* is a Java program that uses at least one Session Initiation Protocol (SIP) servlet.

A SIP servlet is a Java-based application component that is managed by a SIP servlet container and that performs SIP signaling. Like other Java-based components, servlets are platform-independent Java classes that are compiled to platform-neutral bytecode that can be loaded dynamically into and run by a Java-enabled SIP application server. Containers, sometimes called servlet engines, are server extensions that handle servlet interactions. SIP servlets interact with clients by exchanging request and response messages through the servlet container.

SIP is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging. ″Presence″ in this context refers to user status such as ″Active,″ ″Away,″ or ″Do not disturb.″ The standard that defines a programming model for writing SIP-based servlet applications is JSR 116.

SIP container

A *SIP container* is a Web application server component that invokes the Session Initiation Protocol (SIP) action servlet and that interacts with the action servlet to process SIP requests.

The servlet container provides the network services over which requests and responses are received and sent. It decides which applications to invoke and in what order. The container also contains and manages servlets through their life cycle.

A SIP servlet container manages the network listener points on which it listens for incoming SIP traffic. A listener point is a combination of transport protocol, IP address, and port number. The SIP specification (JSR 116) requires all SIP elements to support both UDP and TCP, and optionally TLS, SCTP, and potentially other transports.

Configuring the SIP container

Learn how to configure SIP container timers and custom properties

You can configure and optimize the Session Initiation Protocol (SIP) container for your specific environment. For example, you may want to adjust message response times or set a custom property. The administrative console allows you to modify the default settings for many SIP container properties.

Follow these basic steps to find and configure your SIP container settings:

1. Stop WebSphere Application Server, if it is running.
2. Open the administrative console.
3. Expand the Servers section and click **Application servers** → ***serverName*** to open the configuration tab for your server.
4. Under **Container Settings**, expand the **SIP Container Settings** section and click **SIP Container** to display the SIP container configuration tab. You may now select which container settings you want to change:
   - **General Properties** lets you configure session, message, and response time maximums. See the list of Session Initiation Protocol (SIP) container settings for information about specific properties and settings.
   - The **Additional Properties** section contains links to fields that allow you to define custom properties, configure the session manager, or manage the inbound channel settings. Refer to Configuring transport chains and Session Initiation Protocol (SIP) container inbound channel settings for detailed instructions.
5. After you have entered your configuration changes, click **Apply** and exit the administration console.
6. Start WebSphere Application Server.

Your changes to the SIP container will take effect immediately after you start WebSphere Application Server.

Configuring SIP timers

Information for configuring the Timer B the timeout for receiving INVITE replies setting

SIP timers have an effect on the functional configuration of the product. You can configure Timer B, the timeout in milliseconds for receiving a reply from a client after the SIP container sends an INVITE request to that client.

If a SIP container does not receive a reply from a client after the defined Timer B timeout value, the container sends error message 408 to the application. The default timeout value is 64*T1, which equals 32 000 milliseconds (32 seconds) when Timer T1 is set to its default value of 500 milliseconds. To set the Timer B value:

1. Expand the Servers section and click **Application servers** → ***serverName*** to open the configuration tab for your server.
2. Under **Container Settings**, expand the **SIP Container Settings** section and click **SIP Container** to display the SIP container configuration tab.
3. Under the **Additional Properties** section click **Custom properties**.
4. On the **Custom properties** screen, do one of the following:
   a. If your Timer B property already exists in the list of custom properties, find and click on your Timer B settings to bring up the **Configuration** tab and enter your new setting in the **value** field.
   b. Click **New** to bring up the **Configuration** tab and enter `javax.sip.transaction.timerb` in the **name** field and your desired value in milliseconds in the **value** field.

   Then click **Apply** to make your changes.

5. Click **save** to save your settings to the master configuration. You may need to restart your server for the changes to take effect.

SIP timer summary

Request for Comments (RFC) 3261, “SIP: Session Initiation Protocol,” specifies various timers that SIP uses.

Table 4 summarizes for each SIP timer the default value, the section of RFC 3261 that describes the timer, and the meaning of the timer.

Table 4. Summary of SIP timers

Timer	Default value	Section	Meaning
T1	500 ms	17.1.1.1	Round-trip time (RTT) estimate
T2	4 sec.	17.1.2.2	Maximum retransmission interval for non-INVITE requests and INVITE responses
T4	5 sec.	17.1.2.2	Maximum duration that a message can remain in the network
Timer A	initially T1	17.1.1.2	INVITE request retransmission interval, for UDP only
Timer B	64*T1	17.1.1.2	INVITE transaction timeout timer
Timer D	> 32 sec. for UDP	17.1.1.2	Wait time for response retransmissions
	0 sec. for TCP and SCTP		
Timer E	initially T1	17.1.2.2	Non-INVITE request retransmission interval, UDP only
Timer F	64*T1	17.1.2.2	Non-INVITE transaction timeout timer
Timer G	initially T1	17.2.1	INVITE response retransmission interval
Timer H	64*T1	17.2.1	Wait time for ACK receipt
Timer I	T4 for UDP	17.2.1	Wait time for ACK retransmissions
	0 sec. for TCP and SCTP		
Timer J	64*T1 for UDP	17.2.2	Wait time for retransmissions of non-INVITE requests
	0 sec. for TCP and SCTP		
Timer K	T4 for UDP	17.1.2.2	Wait time for response retransmissions
	0 sec. for TCP and SCTP		

Session Initiation Protocol (SIP) container settings

Use this page to configure the Web container settings.

To view this administrative console page, click **Application servers** → ***serverName*** → **SIP container settings** → **SIP container**.

Related information

Server collection
Use this page to view information about and manage application servers, Java message service (JMS) servers, and Web servers.

Maximum application sessions

Specifies the maximum number of SIP application sessions that the container manages. When the maximum has been reached, no new SIP conversations are started.

Data type	Integer
Default	120000 (recommended)
Range	1 <= n <= java.lang.Integer.MAX_VALUE

Maximum messages per averaging period

Specifies the maximum amount of SIP messages per averaging period.

Data type	Integer
Default	5000 (recommended)
Range	1 <= n <= java.lang.Integer.MAX_VALUE

Maximum queue dispatch size

Specifies the size of the internal dispatch queue. Specifying anything higher than this size overloads the queue. When the internal queue reaches the overloaded state, incoming UDP packets will be dropped until the queue exits the overloaded state. Limiting the queue size enables better recovery from the case where the CPU is used by other processes or threads (for example, garbage collection) and prevents the container from reaching out-of-memory conditions. Setting the value to 0 gives an unlimited queue size.

Data type	Integer
Default	5000 (recommended)
Range	1 <= n <= java.lang.Integer.MAX_VALUE

Maximum response time

Specifies the maximum acceptable response time in milliseconds for an application. When the value of this parameter has exceeded the specified time, the container notifies the clustering framework that it is unavailable.

Use the maxSipResponseTime parameter carefully because the calculated response time does not match the behavior of all applications. For requests, such as INVITE, where the responses are generated as a result of a user interaction (a user picking up a handset, for example), the calculated response time is extensive. However, in this example, the extensive response time is not caused by a delay in the SIP container. Therefore, do not calculate the response time as a load factor. The recommended applications for effective calculation of response time are applications that respond immediately without a user interaction. The Subscribe and Register applications are relevant examples.

Data type	Integer
Default	0
Range	0 <= n <= java.lang.Integer.MAX_VALUE

Averaging period in milliseconds

Specifies that if a servlet is invoked one time while generating output to be cached, a cache entry is created. This entry contains the cache output and the side effects of the invocation. Example of side effects include calls to other servlets or JavaServer Pages (JSP) files, and metadata about the entry, such as timeout and entry priority information.

Data type	Integer
Default	1000 (recommended)
Range	1000 <= n <= java.lang.Integer.MAX_VALUE

Statistic update rate in milliseconds

Specifies control over the interval for which the container calculates averages and publishes statistics to Performance Monitoring Infrastructure.

Data type	Integer
Default	1000 (recommended)
Range	1000 <= n <= java.lang.Integer.MAX_VALUE

Deploying SIP applications

Use the administrative console to customize your Session Initiation Protocol (SIP) application installation

When you deploy a Session Initiation Protocol (SIP) application, you can perform various tasks such as installing, starting, stopping, upgrading, and uninstalling the application.

SIP applications are installed as Java 2 Platform Enterprise Edition (J2EE) applications. You can deploy a SIP application from a graphical interface or from a command line.

Deploying SIP applications through the console

You can deploy a Session Initiation Protocol (SIP) application through the administrative console.

SIP applications are deployed as Java 2 Platform Enterprise Edition (J2EE) applications. In order to process requests, a virtual host must be defined when deploying the SIP application. If there is no virtual host defined for the configured SIP container listen port, the installed application will be inaccessible.

1. Open the administrative console.

   In a browser, go to URL `http://`*`hostname`*`:9090/admin`, where *hostname* is the name of the host computer. Enter the appropriate login information, and click **OK**.
2. In the left frame click **Applications** → **Install New Application**.
3. Browse and select a SAR file. Specify the context root, beginning with a slash (/), in the **Context Root** field. For example, if your application is named ThisApplication, type `/ThisApplication`.
4. Click **Next** (under the **Context Root** field not beside the WebSphere Status title). If the SAR file has been assembled correctly, the screen will still have the title “Preparing for the application installation”, but the content will change. If an error message appears, check the contents of the SAR file; in particular, verify the web.xml file contents, and try to reload the SAR file.
5. Click **Next**. If you see a screen indicating “Application Security Warnings”, click **Continue**.
6. The **Install New Application** screen should appear with “Step 1: Select application options” highlighted. Select the options you need and click **Next**.
7. “Step 2: Map modules to servers” should appear highlighted now. You can choose the cluster or server where you want to install the application’s modules.
   - If you are installing the application in a stand-alone system, click **Next**.
   - If you are installing the application in a clustered system, select **WebSphere:cell=*cellname*,cluster=*cluster_name*** in the **Clusters and Servers** field, select the check box beside the Web module that you want to install, and click **Apply** and **Next**.
8. Now “Step 3: Map virtual hosts for Web modules” should appear highlighted. To the right of the application name there should be a drop-down labeled **Virtual Host**.
   - If you are installing the application in a standalone system, set the value of the drop-down to **default_host**, and click **Next**.
   - If you are installing the application in a clustered system, set the value of the drop-down to the name of the virtual host that was chosen during setup, and click **Next**.

   Remember: You must define a virtual host for your configured SIP container listen port or else you will not be able to access the application.
9. You should now see “Step 4: Summary” highlighted. In the right panel you will see a **Summary of installation options** table that details your selected options and their values. If you need to change an option, click **Previous** to return to the section where you can make your change. Click **Finish** to

install the application with your settings. The screen should display, `Application` *appname*`_sar installed successfully`, where *appname* is the name of the application.

10. Click the **Save to Master Configuration** link. A Save to Master Configuration window appears.
11. In the Save to Master Configuration window, click **Save**. The application has now been saved in the current configuration.
12. To confirm that the installation succeeded, in the left frame click **Applications** → **Enterprise Applications**. The newly installed application should appear in the list of installed applications as *appname*_sar.
13. To start the application so that it can service SIP requests, check the box beside *appname*_sar, and click **Start**. You might also want to look at the logs for a successful startup message.

The application can service SIP requests now.

Deploying SIP applications through scripting

You can deploy a Session Initiation Protocol (SIP) application not only from the GUI but also from the command line.

- Launch a scripting client. For more information, see AdminApp object for scripted administration.
- List applications.
- Install standalone archive files. For more information about installation, see Installation options for the AdminApp object.
- Edit application configurations.
- Uninstall applications.

Chapter 10. EJB applications

Task overview: Using enterprise beans in applications

This article provides an overview of the tasks you must perform to use enterprise beans in a J2EE application.

1. Design a J2EE application and the enterprise beans that it needs. For links to design information that is specific to enterprise beans, see “Data access: Resources for learning” on page 583.
2. Develop any enterprise beans that your application will use.
3. Prepare for assembly. For your EJB 2.x-compliant entity beans, decide on an appropriate access intent policy.
4. Assemble the beans into one or more EJB modules using one of the assembly tools. This process includes setting security. For your EJB 2.x-compliant entity beans, you might also want to designate container-managed persistence (CMP) sequence groups.
5. Assemble the modules into a J2EE application using the assembly tool.
6. For a given application server, update the EJB container configuration if needed for the application to be deployed, and determine if you want to batch commands or defer commands for container-managed persistence.
7. Deploy the application in an application server.
8. Test the modules.
   - As needed, debug problems with the container.
   - Debug access problems.
9. Assemble the production application using one of the assembly tools
10. Deploy the application to a production environment.
11. Manage the application:
    a. Manage installed EJB modules. After an application has been installed, you can manage its EJB modules individually through the Assembly Service Toolkit.
    b. Manage other aspects of the J2EE application.
12. Update the module and redeploy it using one of the assembly tools.
13. Tune the performance of the application. See Best practices for developing enterprise beans.

Enterprise beans

An enterprise bean is a Java component that can be combined with other resources to create J2EE applications. There are three types of enterprise beans, *entity* beans, *session* beans, and *message-driven* beans.

All beans reside in EJB containers, which provide an interface between the beans and the application server on which they reside.

Entity beans store permanent data, so they require connections to a form of persistent storage. This storage might be a database, an existing legacy application, a file, or another type of persistent storage.

Session beans typically contain the high-level and mid-level business logic for an application. Each method on a session bean typically performs a particular high-level operation. For example, submitting an order or transferring money between accounts. Session beans often invoke methods on entity beans in the course of their business logic.

Session beans can be either *stateful* or *stateless*. A stateful bean instance is intended for use by a single client during its lifetime, where the client performs a series of method calls that are related to each other in time for that client. One example is a ″shopping cart″ where the client adds items to the cart over the course of an online shopping session. In contrast, a stateless bean instance is typically used by many

clients during its lifetime, so stateless beans are appropriate for business logic operations that can be completed in the span of a single method invocation. Stateful beans should be used only where absolutely necessary -- using stateless beans improves the ability to debug, maintain, and scale the application.

Message-driven beans enable asynchronous message servicing.

- The EJB container and a Java Message Service (JMS) provider work together to process messages. When a message arrives from another application component through JMS, the EJB container forwards it through an onMessage() call to a message-driven bean instance, which then processes the message. In other respects, message-driven beans are similar to stateless session beans.
- The EJB container and a Java Connector Architecture (JCA) resource adapter work together to process messages from an enterprise information system (EIS). When a message arrives from an EIS, the resource adapter receives the message and forwards it to a message-driven bean, which then processes the message. The message-driven bean is provided services such as transaction support by the EJB container in the same way that other enterprise beans are provided service.

Beans that require data access use *data sources*, which are administrative resources that define pools of connections to persistent storage mechanisms.

For more information about enterprise beans, see “Enterprise beans: Resources for learning” on page 197.

EJB modules

An EJB module is used to assemble one or more enterprise beans into a single deployable unit. An EJB module is stored in a standard Java archive (JAR) file.

An EJB module contains the following:

- One or more deployable enterprise beans.
- A deployment descriptor, stored in an Extensible Markup Language (XML) file. This file declares the contents of the module, defines the structure and external dependencies of the beans in the module, and describes how the beans are to be used at run time.

You can deploy an EJB module as a stand alone application, or combine it with other EJB modules or with Web modules to create a J2EE application. An EJB module is installed and run in an enterprise bean container.

For more information about EJB modules, see “Enterprise beans: Resources for learning” on page 197

EJB containers

An Enterprise JavaBeans (EJB) container provides a run-time environment for enterprise beans within the application server. The container handles all aspects of an enterprise bean’s operation within the application server and acts as an intermediary between the user-written business logic within the bean and the rest of the application server environment.

One or more EJB modules, each containing one or more enterprise beans, can be installed in a single container.

The EJB container provides many services to the enterprise bean, including the following:

- Beginning, committing, and rolling back transactions as necessary.
- Maintaining pools of enterprise bean instances ready for incoming requests and moving these instances between the inactive pools and an active state, ensuring that threading conditions within the bean are satisfied.
- Most importantly, automatically synchronizing data in an entity bean’s instance variables with corresponding data items stored in persistent storage.

By dynamically maintaining a set of active bean instances and synchronizing bean state with persistent storage when beans are moved into and out of active state, the container makes it possible for an application to manage many more bean instances than could otherwise simultaneously be held in the application server's memory. In this respect, an EJB container provides services similar to virtual memory within an operating system.

By default, an EJB container runs in the **quick start** mode. The EJB container startup logic delays the loading and processing of all EJB types *except* Message Driven Beans (because they must exist before messages are posted for them), Startup Beans (which must be processed at server startup time), and those EJB types that you specify to initialize at server start. For more information about disabling quick start for EJB types, see "Changing enterprise bean types to initialize at application start time using the Application Server Toolkit" on page 220.

All other EJB initialization is delayed until the first use of the EJB type. When using Local Interfaces, the first use is when you perform an *InitialContext.lookup()* method for the type. For Remote Interfaces, it is when you call the first method on an EJB or its Home.

For more information about EJB containers, see "Enterprise beans: Resources for learning."

Enterprise beans: Resources for learning

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

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

Planning, business scenarios, and IT architecture

- Mastering Enterprise JavaBeans

  A comprehensive treatment of Enterprise JavaBeans (EJB) programming in nonprintable form (PDF). One must be registered to download the PDF, but registration is free. Information about purchasing a hardcopy is available on the Web site.
- *Enterprise JavaBeans* by Richard Monson-Haefel (O'Reilly and Associates, Inc.: Third Edition, 2001)

Programming model and decisions

- Read all about EJB 2.0

  A comprehensive overview of the 2.0 specification that is still relevant to users of EJB 2.1.
- The J2EE Tutorial

  This set of articles by Sun Microsystems covers several EJB-related topics, including the basic programming models, persistence, and EJB Query Language.

Programming instructions and examples

- WebSphere Application Server Development Best Practices for Performance and Scalability

  Programming practice for enterprise beans and other types of J2EE components.
- Optimistic Locking in IBM WebSphere Application Server 4.0.2

  Examples of the effect of optimistic concurrency on application behavior. Although the paper is based on a previous version of this product, the data access issues discussed in it are current.

  This paper does not seem to be available directly by URL. To view this paper, visit the specified URL and search on `"optimistic locking"`

Programming specifications

- Enterprise JavaBeans 2.1 Specification

  You can download the specification from this URL.
- Enterprise JavaBeans 3.0 Specification

  You can download the specification from this URL.
- Java™ 2 Platform: Compatibility with Previous Releases

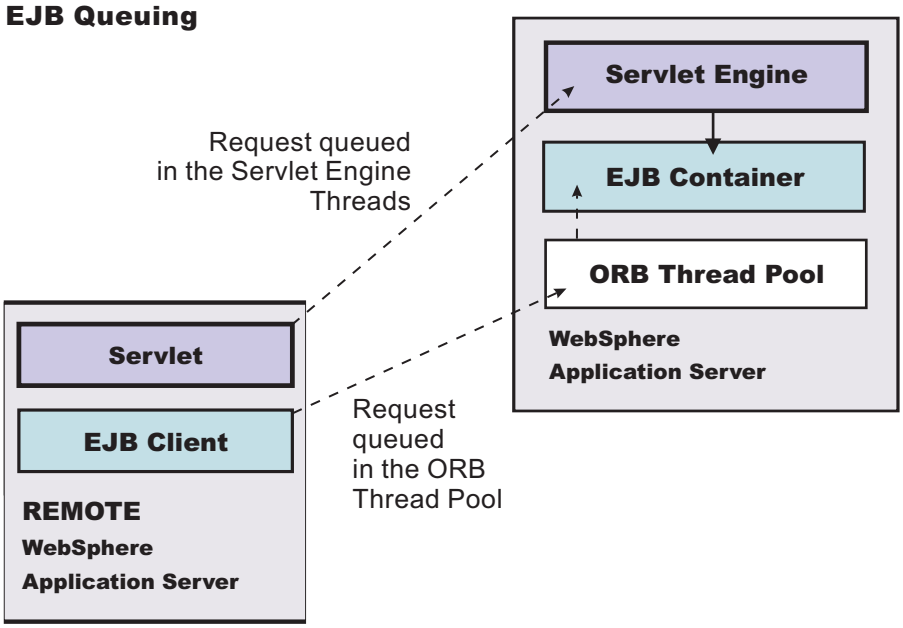
  This Sun Microsystems article includes both source and binary compatibility issues.

EJB method Invocation Queuing

Method invocations to enterprise beans are only queued for remote clients making the method call. An example of a remote client is an enterprise Java bean (EJB) client running in a separate Java virtual machine (JVM) (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client, either a servlet or another enterprise bean, is installed in the same JVM on which the EJB method runs and on the same thread of execution as the EJB client.

Remote enterprise beans communicate by using the Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP). Method invocations initiated over RMI-IIOP are processed by a server-side object request broker (ORB). The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, due to the use of unbounded threads.

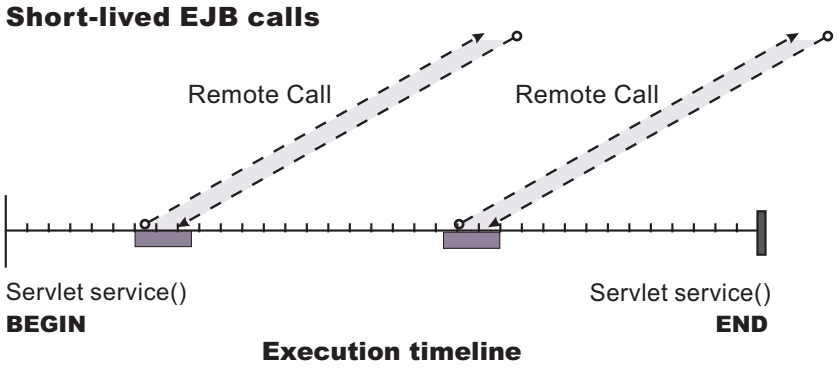
The following illustration depicts the two queuing options of enterprise beans.

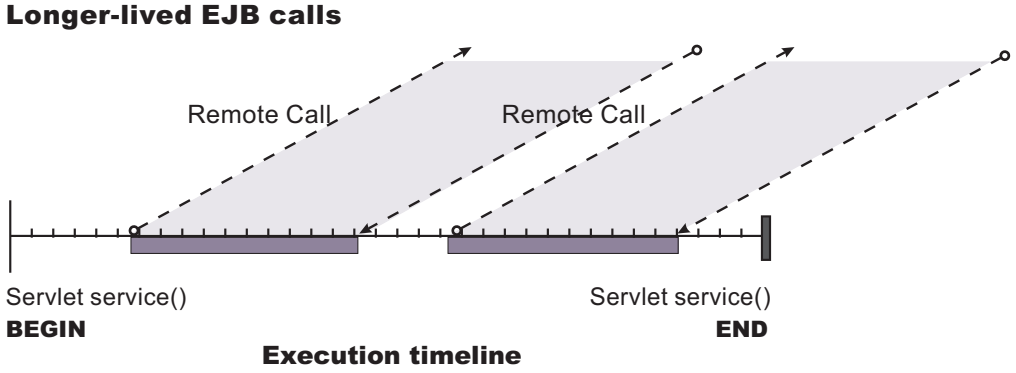


The following are two tips for queueing enterprise beans:

- **Analyze the calling patterns of the EJB client**.

  When configuring the thread pool, it is important to understand the calling patterns of the EJB client. If a servlet is making a small number of calls to remote enterprise beans and each method call is relatively quick, consider setting the number of threads in the ORB thread pool to a value lower than the Web container thread pool size value.





The degree to which the ORB thread pool value needs increasing is a function of the number of simultaneous servlets, that is, clients, calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

- **Monitor the percentage of configured threads in use**.

  Tivoli Performance Viewer shows a metric called *percent maxed*, which is used to determine how often the configured threads are used. A value that is consistently in the double-digits, indicates a possible bottleneck a the ORB. Increase the number of threads.

See also Queuing network

Enterprise bean and EJB container troubleshooting tips

If you are having problems starting an EJB container, or encounter error messages or exceptions that appear to be generated on by an EJB container, follow these steps to resolve the problem:

- Use the Administrative Console to verify that the application server which hosts the container is running.
- Browse the JVM log files for the application server which hosts the container. Look for the message **server *server_name* open for e-business** in the `SystemOut.log` . If it does not appear, or if you see the message **problems occurred during startup**, browse the `SystemErr.log` for details.
- Browse the system log files for the application server which hosts the container.
- Enable tracing for the EJB Container component, by using the following trace specification `EJBContainer=all=enabled`. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

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 IBM Support page.

Error in client log: Missing jar file

The following error message appears in the client log file because a JAR file is missing from the classpath on the client machine. The Object Request Broker (ORB) needs this file to unmarshal the nested exception that is part of the EJB exception, returned by the server to the client application. For example, if the EJB returns a DB2® JCC SQL exception nested inside of the EJB exception that it returns to the client, the ORB is not able to unmarshal the nested exception if the db2jcc.jar file that contains the DB2 SQL exception is not in the client classpath.

```
java.rmi.MarshalException: CORBA MARSHAL 0x4942f89a No; nested exception is:
org.omg.CORBA.MARSHAL: Unable to read value
from underlying bridge : Custom marshaling (4) Sender's class does not match
local class  vmcid: 0x4942f000  minor  code: 2202  completed: No*
```

To avoid this error, include the JAR file that contains the class for the nested exception that is returned in the EJB exception.

Cannot access an enterprise bean from a servlet, a JSP file, a stand-alone program, or another client

This article provides troubleshooting tips for problems related to accessing enterprise beans.

What kind of error are you seeing?

- **javax.naming.NameNotFoundException: Name *name* not found in context ″local″ message** when access is attempted
- **BeanNotReentrantException** is thrown
- **CSITransactionRolledbackException / TransactionRolledbackException** is thrown
- Call fails, Stack trace beginning **EJSContainer E Bean method threw exception [exception_name]** found in JVM log file.
- Call fails, **ObjectNotFoundException or ObjectNotFoundLocalException** when accessing stateful session EJB found in JVM log file.
- Attempt to start CMP EJB module fails with **javax.naming.NameNotFoundException: *dataSourceName***
- **Transaction [tran ID] has timed out after 120 seconds** error accessing EJB.
-
- Symptom: **CNTR0001W: A Stateful SessionBean could not be passivated**
- Symptom: **org.omg.CORBA.BAD_PARAM: Servant is not of the expected type. minor code: 4942F21E completed: No** returned to client program when attempting to execute an EJB method

If the client is remote to the enterprise bean, which means, running in a different application server or as a stand-alone client, browse the JVM logs of the application server hosting the enterprise bean as well as log files of the client.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, perform these steps:

1. If the problem appears to be name-service related, which means that you see a NameNotFoundException, or a message ID beginning with NMSV, see these topics for more information:
   - Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client
   - Naming service troubleshooting tips
2. Check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

If you still cannot fix your problem, seeTroubleshooting help from IBM for further assistance.

ObjectNotFoundException or ObjectNotFoundLocalException when accessing stateful session EJB

A possible cause of this problem is that the stateful session bean timed out and was removed by the container. This event must be addressed in the code, according to the EJB 2.1 specification (available at `http://java.sun.com/products/ejb/docs.html`), section 7.6.2, Dealing with exceptions.

Stack trace beginning ″EJSContainer E Bean method threw exception [exception_name]″ found in JVM log file

If the exception name indicates an exception thrown by an IBM class that begins with ″com.ibm...″, then search for the exception name within the information center, and in the online help as described below. If ″exception name″ indicates an exception thrown by your application, contact the application developer to determine the cause.

javax.naming.NameNotFoundException: Name name not found in context ″local″

A possible reason for this exception is that the enterprise bean is not local (not running in the same Java virtual machine [JVM] or application server) to the client JSP, servlet, Java application, or other enterprise bean, yet the call is to a ″local″ interface method of the enterprise bean . If access worked in a development environment but not when deployed to WebSphere Application Server, for example, it might be that the enterprise bean and its client were in the same JVM in development, but are in separate processes after deployment.

To resolve this problem, contact the developer of the enterprise bean and determine whether the client call is to a method in the local interface for the enterprise bean. If so, have the client code changed to call a remote interface method, or to promote the local method into the remote interface.

References to enterprise beans with local interfaces are bound in a name space local to the server process with the URL scheme of `local:`. To obtain a dump of a server `local:` name space, use the name space dump utility described in the article ″Name space dump utility for `java:`, `local:` and `server` name spaces.″

BeanNotReentrantException is thrown

This problem can occur because client code (typically a servlet or JSP file) is attempting to call the same stateful SessionBean from two different client threads. This situation often results when an application stores the reference to the stateful session bean in a static variable, uses a global (static) JSP variable to refer to the stateful SessionBean reference, or stores the stateful SessionBean reference in the HTTP session object. The application then has the client browser issue a new request to the servlet or JSP file before the previous request has completed.

To resolve this problem, ask the developer of the client code to review the code for these conditions.

CSITransactionRolledbackException / TransactionRolledbackException is thrown

An enterprise bean container creates these high-level exceptions to indicate that an enterprise bean call could not successfully complete. When this exception is thrown, browse the JVM logs to determine the underlying cause.

Some possible causes include:
- The enterprise bean might throw an exception that was not declared as part of its method signature. The container is required to roll back the transaction in this case. Common causes of this situation are where the enterprise bean or code that it calls creates a NullPointerException, ArrayIndexOutOfBoundsException, or other Java runtime exception, or where a BMP bean encounters a JDBC error. The resolution is to investigate the enterprise bean code and resolve the underlying exception, or to add the exception to the problem method signature.
- A transaction might attempt to do additional work after being placed in a ″Marked Rollback″, ″RollingBack″, or ″RolledBack″ state. Transactions cannot continue to do work after they are set to one of these states. This situation occurs because the transaction has timed out which, often occurs because of a database deadlock. Work with the application database management tools or administrator to determine whether database transactions called by the enterprise bean are timing out.
- A transaction might fail on commit due to dangling work from local transactions. The local transaction encounters some ″dangling work″ during commit. When a local transactions encounters an ″unresolved action″ the default action is to ″rollback″. You can adjust this action to ″commit″ in an assembly tool. Open the enterprise bean .jar file (or the EAR file containing the enterprise bean) and select the Session Beans or Entity Beans object in the component tree on the left. The Unresolved Action property is on the IBM Extensions tab of the container properties.

Attempt to start EJB module fails with ″javax.naming.NameNotFoundException dataSourceName_CMP″exception

This problem can occur because:
- When the DataSource resource was configured, container managed persistence was not selected.
  - To confirm this problem, in the administrative console, browse the properties of the data source given in the NameNotFoundException. On the Configuration panel, look for a check box labeled **Container Managed Persistence**.
  - To correct this problem, select the check box for **Container Managed Persistence**.
- If container managed persistence is selected, it is possible that the CMP DataSource could not be bound into the namespace.
  - Look for additional naming warnings or errors in the status bar, and in the hosting application server JVM logs. Check any further naming-exception problems that you find by looking at the topic Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client.

Transaction [tran ID] has timed out after 120 seconds accessing an enterprise bean

This error can occur when a client executes a transaction on a CMP or BMP enterprise bean.
- The default timeout value for enterprise bean transactions is 120 seconds. After this time, the transaction times out and the connection closes.
- If the transaction legitimately takes longer than the specified timeout period, go to **Manage Application Servers** > ***server_name***, select the **Transaction Service properties** page, and look at the property **Total transaction lifetime timeout**. Increase this value if necessary and save the configuration.

Symptom:CNTR0001W: A Stateful SessionBean could not be passivated

This error can occur when a Connection object used in the bean is not closed or nulled out.

To confirm this is the problem, look for an exception stack in the JVM log for the EJB container that hosts the enterprise bean, and looks similar to:

```
[time EDT] <ThreadID> StatefulPassi W CNTR0001W:
A Stateful SessionBean could not be passivated: StatefulBean0
(BeanId(XXX#YYY.jar#ZZZZ, <ThreadID>),
state = PASSIVATING) com.ibm.ejs.container.passivator.StatefulPassivator@<ThreadID>
java.io.NotSerializableException: com.ibm.ws.rsadapter.jdbc.WSJdbcConnection
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at java.io.ObjectOutputStream.outputClassFields((Compiled Code))
 at java.io.ObjectOutputStream.defaultWriteObject((Compiled Code))
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at com.ibm.ejs.container.passivator.StatefulPassivator.passivate((Compiled Code))

 at com.ibm.ejs.container.StatefulBean0.passivate((Compiled Code)
 at com.ibm.ejs.container.activator.StatefulASActivationStrategy.atUnitOfWorkEnd
                      ((Compiled Code))
 at com.ibm.ejs.container.activator.Activator.unitOfWorkEnd((Compiled Code))
 at com.ibm.ejs.container.ContainerAS.afterCompletion((Compiled Code)
```

where *XXX,YYY,ZZZ* is the Bean's name, and *<ThreadID>* is the thread ID for that run.

To correct this problem, the application must close all connections and set the reference to null for all connections. Typically this activity is done in the `ejbPassivate()` method of the bean. See the enterprise bean specification mandating this requirement, specifically section 7.4 in the EJB specification Version 2.1. Also, note that the bean must have code to reacquire these connections when the bean is reactivated. Otherwise, there are NullPointerExceptions when the application tries to reuse the connections.

Symptom: org.omg.CORBA.BAD_PARAM: Servant is not of the expected type. minor code: 4942F21E completed: No

This error can be returned to a client program when the program attempts to execute an EJB method.

Typically this problem is caused by a mismatch between the interface definition and implementation of the client and server installations, respectively.

Another possible cause is when an application server is set up to use a single class loading scheme. If an application is uninstalled while the application server remains active, the classes of the uninstalled application are still loaded in the application server. If you change the application, redeploy and reinstall it on the application server, the previously loaded classes become back level. The back level classes cause a code version mismatch between the client and the server.

To correct this problem:
1. Change the application server class loading scheme to multiple.
2. Stop and restart the application server and try the operation again.
3. Make sure the client and server code version are the same.

Using access intent policies

You can use access intent policies to help the product runtime environment manage various aspects of Enterprise JavaBeans (EJB) persistence.

You apply access intent policies to EJB Version 2.0 (and later) entity beans and their methods by using an application assembly tool. A set of default access intent policies comes with the Application Server Toolkit (AST) .

1. Apply default access intent to CMP entity beans. For more information, see the online help available with the Application Server Toolkit.
2. Apply access intent policies to methods of CMP entity beans.

Access intent policies

An access intent policy is a named set of properties (access intents) that governs data access for Enterprise JavaBeans (EJB) persistence. You can assign policies to an entity bean and to individual methods on an entity bean's home, remote, or local interfaces during assembly. You can set access intents only within EJB Version 2.x-compliant modules for entity beans with CMP Version 2.x.

This product supplies a number of access intent policies that specify permutations of read intent and concurrency control; the pessimistic/update policy can be qualified further. The selected policy determines the appropriate isolation level and locking strategy used by the run time environment.

transition: Access intent policies are specifically designed to supplant the use of isolation level and access intent method-level modifiers found in the extended deployment descriptor for EJB version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB version 2.x enterprise beans.

Access intent policies configured on an entity basis define the default access intent for that entity. The default access intent controls the entity unless you specify a different access intent policy based on either method-level configuration or application profiling.

Note: Method level access intent has been deprecated for Version 6.

You can use application profiling or method level access intent policies to control access intent more precisely. Method-level access intent policies are named and defined at the module level. A module can have one or many such policies. Policies are assigned, and apply, to individual methods of the declared interfaces of entity beans and their associated home interfaces. A method-based policy is acted upon by the combination of the EJB container and persistence manager when the method causes the entity to load.

For entity beans that are backed by tables with nullable columns, use an optimistic policy with caution. The top down default mapping excludes nullable fields. You can override this when doing a meet-in-middle mapping. The fields used in overqualified updates are specified in the ejb-rdb mapping. If nullable columns are selected as overqualified columns, then partial update should also be selected.

Note: When using DB2 for z/OS Version 8, nullable OCC columns create no problems. This is true for JDBC and SQLJ deploy options, and partial and full update.

An entity that is configured with a read-only policy that causes a bean to be activated can cause problems if updates are attempted within the same transaction. Those changes are not committed, and the process throws an exception because data integrity might be compromised.

Concurrency control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction.

The objective of optimistic concurrency is to minimize the time over which a given resource would be unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

Under an optimistic scheme, locks are obtained immediately before a read operation and released immediately afterwards. Update locks are obtained immediately before an update operation and held until the end of the transaction.

To enable optimistic concurrency, this product uses an *overqualified update scheme* to test whether the underlying data source has been updated by another transaction since the beginning of the current

transaction. With this scheme, the columns marked for update and their original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back: All work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection.

best-practices: Whether or not to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. (A high-penalty transaction is one for which recovery would be risky or resource-intensive.) For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme. In general, optimistic concurrency is more efficient when update collisions are expected to be infrequent; pessimistic concurrency is more efficient when update collisions are expected to occur often.

Read-ahead hints

Read-ahead schemes enable applications to minimize the number of database round trips by retrieving a working set of container-managed persistence (CMP) beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that an application most likely needs next. A *read-ahead hint* is a representation of the related beans to read. The hint is associated with the *findByPrimaryKey* method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean.

A read-ahead hint takes the form of a character string. You do not have to provide the string; the wizard generates it for you based on the container-managed relationships (CMRs) that are defined for the bean. The following example is provided as supplemental information only. Suppose a CMP bean type A has a finder method that returns instances of bean A. A read-ahead hint for this method is specified using the following notation: `RelB.RelC; RelD`

Interpret the preceding notation as follows:
- Bean type A has a CMR with bean types B and D.
- Bean type B has a CMR with bean type C.

For each bean of type A that is retrieved from the database, its directly-related B and D beans and its indirectly-related C beans are also retrieved. The order of the retrieved bean data columns in each row of the result set is the same as the order in the read-ahead hint: an A bean, a B bean (or null), a C bean (or null), a D bean (or null). For hints in which the same relationship is mentioned more than once (for example, `RelB.RelC;RelB.RelE`), the data columns for a bean occur only once in the result set, at the position the bean first occupies in the hint.

The tokens shown in the notation (*RelB* and so on) must be CMR field names for the relationships, as defined in the deployment descriptor for the bean. In indirect relationships such as `RelB.RelC`, *RelC* is a CMR field name that is defined in the deployment descriptor for bean type B.

A single read-ahead hint cannot refer to the same bean type in more than one relationship. For example, if a Department bean has an *employees* relationship with the Employee bean and also has a *manager* relationship with the Employee bean, the read-ahead hint cannot specify both *employees* and *manager*.

For more information about how to set read-ahead hints, see the documentation for the Rational Application Developer product.

Run-time behaviors of read-ahead hints

When developing your read-ahead hints, consider the following tips and limitations:

- Read-ahead hints on long or complex paths can result in a query that is too complex to be useful. Read-ahead hints on root or leaf inheritance mappings need particular care. Add up the number of tables that potentially comprise a read-ahead preload to gauge the complexity of the join operations that are required. Consider if the resulting statement constitutes a reasonable query on your target database.
- Read-ahead hints do not work in the following cases:
  - Preload paths across M:N relationships
  - Preload paths across recursive enterprise bean relationships or recursive fk relationships
  - When a read-head hint applies to a SELECT FOR UPDATE statement that requires a table join in a database that does not support the combination of those two operations.

    Generally, the persistence manager issues a SELECT FOR UPDATE statement for a bean only if the bean has an access intent that enforces strict locking policies. Strict locking policies require SELECT FOR UPDATE statements for database select queries. If the database table design requires a join operation to fulfill the statement, many databases issue exceptions because these databases do not support table joins with SELECT FOR UPDATE statements. In those cases, WebSphere Application Server does not implement a read-ahead hint. If the database does provide that support, Application Server implements the read-ahead hints that you configure.

    DB2 Universal Database V8.2 supports SELECT FOR UPDATE statements with table joins.
  - When a read-ahead hint contains a table join

    Different access intents can result in requiring a SELECT FOR UPDATE statement. Check the matrix on the JDBC driver and SELECT FOR UPDATE support to see if readAhead is enabled.

Database deadlocks caused by lock upgrades

To avoid databse deadlocks caused by lock upgrades, you can change the access intent policy for entity beans from the default of wsPessimisticUpdate-WeakestLockAtLoad to wsPessimisticUpdate or can use an optimistic locking approach.

When accessing data in a database concurrently, an application must be aware of and prepared for database locking that must occur to insure the integrity of the data.

If an entity bean performs a findByPrimaryKey (which by default obtains a 'Read' lock in the database), and the entity bean is updated within the same transaction, then a lock upgrade (to 'Exclusive') occurs.

If this scenario occurs on multiple threads concurrently, then a deadlock can happen. This is because multiple 'Read' locks can be obtained concurrently, but one 'Exclusive' lock can be obtained only when all other locks have been dropped. Because all transactions are attempting the lock upgrade in this scenario, this one 'Exclusive' lock can never be obtained .

To avoid this problem, you can change the access intent policy for the entity bean from the default of wsPessimisticUpdate-WeakestLockAtLoad to wsPessimisticUpdate. This change in access intent enables the application to inform WebSphere and the database that the transaction will update the enterprise bean, and so an 'Update' lock is obtained immediately on the findByPrimaryKey. This avoids the lock upgrade when the update is performed later.

The preferred technique to define access intent policies is to change the access intent for the entire entity bean. You can change the access intent for the findByPrimaryKey method, but this is deprecated in Version 6.0. (You might want to change the access intent for an individual method if, for example, the entity bean is involved in some transactions that are read only.)

An alternative technique is to use an optimistic approach, where the findByPrimaryKey method does not hold a 'Read' lock, so there is no lock upgrade. However, this requires that the application is coded for this, to handle rollbacks that could occur. Optimistic locking is really intended for applications that do not expect database contention on a regular basis.

To change the access intent policy for an entity bean, you can use the assembly tool to set the ″Default Access Intent for Entities 2.x (Bean Level)″ on the Access tab of the EJB Deployment Descriptor, as described in "Applying access intent policies to beans" on page 209.

Access intent assembly settings

Access intent policies contain data-access settings for use by the persistence manager. Default access intent policies are configured on the entity bean.

These settings are applicable only for EJB 2.x-compliant entity beans that are packaged in EJB 2.x-compliant modules. Connection sharing between beans with bean-managed persistence and those with container-managed persistence is possible if they all use the same access intent policy.

Name:

Specifies a name for a mapping between an access intent policy and one or more methods.

Description:

Contains text that describes the mapping.

Methods - Name:

Specifies the name of an enterprise bean method, or the asterisk character (*). The asterisk is used to denote all of the methods of an enterprise bean's remote and home interfaces.

Methods - Enterprise bean:

Specifies which enterprise bean contains the methods indicated in the Name setting.

Methods - Type:

Used to distinguish between a method with the same signature that is defined in both the home and remote interface. Use `Unspecified` if an access intent policy applies to all methods of the bean.

Data type	String
Range	Valid values are `Home`, `Remote`,`Local`, `LocalHome` or `Unspecified`

Methods - Parameters:

Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

Applied access intent:

Specifies how the container must manage data access for persistence. Configurable both as a default access intent for an entity and as part of a method-level access intent policy.

Data type	String
Default	`wsPessimisticUpdate-WeakestLockAtLoad`. With Oracle, this is the same as wsPessimisticUpdate.

Range

Valid settings are `wsPessimisticUpdate`, `wsPessimisticUpdate-NoCollision`, `wsPessimisticUpdate-Exclusive`, `wsPessimisticUpdate-WeakestLockAtLoad`, `wsPessimisticRead`, `wsOptimisticUpdate`, or `wsOptimisticRead`. Only `wsPessimisticRead` and `wsOptimisticRead` are valid when class-level caching is enabled in the EJB container.

This product supports lazy collections. For each segment of a collection, iterating through the collection (*next()*) does not trigger a remote method call to retrieve the next remote reference. Two policies (`wsPessimisticUpdate` and `wsPessimisticUpdate-Exclusive`) are extremely lazy; the collection increment size is set to 1 to avoid overlocking the application. The other policies have a collection increment size of 25.

If an entity is not configured with an access intent policy, the run-time environment typically uses `wsPessimisticUpdate-WeakestLockAtLoad` by default. If, however, the **Lifetime in cache** property is set on the bean, the default value of **Applied access intent** is `wsOptimisticRead`; updates are not permitted.

Additional information about valid settings follows:

Profile name	Concurrency control	Access type	Transaction isolation
wsPessimisticRead (Note 1)	pessimistic	read	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate (Note 2)	pessimistic	update	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate-Exclusive (Note 3)	pessimistic	update	serializable
wsPessimisticUpdate-NoCollision (Note 4)	pessimistic	update	read committed
wsPessimisticUpdate-WeakestLockAtLoad (Note 5)	pessimistic	update	Repeatable read
wsOptimisticRead	optimistic	read	read committed
wsOptimisticUpdate (Note 6)	optimistic	update	read committed

Notes:

1. Read locks are held for the duration of the transaction.
2. The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction.
3. SELECT FOR UPDATE is generated; locks are held for the duration of the transaction.
4. A plain SELECT query is generated. No locks are held, but updates are permitted. Use cautiously. This intent enables execution without concurrency control.
5. Where supported by the backend, the generated SELECT query does not include `FOR UPDATE`; locks are escalated by the persistent store at storage time if updates were made. Otherwise, the same as wsPessimisticUpdate.
6. Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction.

   Be sure to review the rules for forming overqualified-update query predicates. Certain column types (for example, BLOB) are ineligible for inclusion in the overqualified-update query predicate and might affect your design.

Access intent for both entity bean types

Container-managed persistence (CMP) developers can use *access intent* to provide hints on how the application server run time should manage the details of persistence without having to explicitly manage any of the persistence logic from within their application.

Using the access intent service is also an option for programmers who develop bean-managed persistence (BMP) entity beans. Because the only meaningful difference between BMP and CMP components is the mechanism that provides the persistence logic, BMP beans leverage access intent hints in the same manner as the EJB container manages accent intent for CMP beans. This ability becomes especially important when BMP entities and CMP entities want to share connections. BMP beans configured with the same concurrency as the CMP beans and implemented to the same isolation level mapping as the CMP can share connections.

Developers can apply access intent policies to BMP entity beans as well as to CMP entity beans. It is expected that BMP developers use only those access intent attributes that are important to a particular BMP bean. The access intent service interface is bound into the *java:comp namespace* for each particular BMP bean. The access intent policy retrieved from the access intent service is current from the time that the *ejbLoad* process is called until the time that the *ejbStore* process completes its invocation.

Applying access intent policies to beans

You can apply an access intent policy to an application's entity beans through the assembly tool.

Note: This is the preferred technique to define access intent policies. Method-level access intent is deprecated in Version 6.0.

1. Start the Application Server Toolkit.
2. **Optional:** Open the J2EE perspective to work with J2EE projects. Click **Window > Open Perspective > Other > J2EE**.
3. **Optional:** Open the Project Explorer view. Click **Window > Show View > Project Explorer**. Another helpful view is the Navigator view (**Window > Show View > Navigator**).
4. Create a new application EAR file or edit an existing one.

   For example, to change attributes of an existing application, use the import wizard to import an EAR file. To start the import wizard:

   a. Select **File > Import > EAR file > Next**
   b. Select the EAR file.
   c. Create a WebSphere Application Server v6.0 type of Server Runtime. Select **New** to open the New Server Runtime Wizard and follow the instructions.
   d. In the *Target server* field, select *WebSphere Application Server v6.0* type of Server Runtime.
   e. Select **Finish**
5. In the Project Explorer view of the J2EE perspective, right-click **Deployment Descriptor: EJB Module Name** under the EJB module for the bean instance, then select **Open With > Deployment Descriptor Editor**. A property dialog notebook for the EJB project is displayed in the property pane.
6. Select the **Access** tab.
7. In the **Access Intent for Entities 2.x (Bean Level)** panel, select the name of the bean.
8. On the right side of the **Access Intent for Entities 2.x (Method Level)** panel, select **Add**. The **Add Access Intent** panel displays.
9. In the **Access intent name** field, select wsPessimisticUpdate from the drop-down list.
10. **Optional:** Enter a **Description** to help you remember what this policy does.
11. **Optional:** Change the **Persistence Option** setting
12. Click **Finish**. The access intent policy for the entity bean is shown in the **Access Intent for Entities 2.x (Bean Level)** panel

Configuring read-read consistency checking with the assembly tools

Read-read consistency checking only applies to LifeTimeInCache beans whose data is read from another transaction. For the Access Intents that are for *repeatable read* (RR), this means the product checks that the data is consistent with that in the data store, and ensures that no one updates it after the checking.

For the Access Intents that are for *read committed* (RC), this means the product checks that the data is consistent at the point of checking, it does **not** guarantee that the data does not change after the checking. This makes the behavior of the LifeTimeInCache bean the same as non-LifeTimeInCache beans.

To perform this checking, you need to configure CMP entity beans with read-read consistency checking. You can do this using the Application Server Toolkit.

1. Start the Application Server Toolkit.
2. In the Project Explorer view of the J2EE perspective, right-click the **Deployment Descriptor: EJB Module Name** under the EJB module for the bean instance, then select **Open With > Deployment Descriptor Editor**. A property dialog notebook for the EJB project is displayed in the property pane.
3. Select the **Access** tab. The Add Access Intent window appears. There are two areas of the panel that deal with adding access intent:
   - Default Access Intent for Entities 2.x (Bean Level)
   - Access Intent for Entities 2.x (Method Level)
4. Select the Bean or Method level. Another access intent window appears where you can set the properties you wish to use.
5. Use the dropdown list to select the Access intent name.
6. **Optional:** Enter a description.
7. Check the **Persistence Option** box.
8. Check the **Verify Read Only Data** box.
9. Use the dropdown list to select your choice for read-read consistency checking. You have three options:

   NONE No read-read checking is done.

   AT_TRAN_BEGIN
   During ejbLoad, if the data is from cache, check the database to ensure that the data of the bean (with proper locking based on access intent's concurrency control attribute) has not changed since the last load.

   AT_TRAN_END
   At the end of transaction, if the bean is not changed and did not load by the current transaction, check the database to ensure that the data of the bean has not changed from last load (with proper locking based on access intent's concurrency control attribute.) If the data has changed, fail the transaction.
10. Select **Finish**.

Examples: read-read consistency checking

Read-read consistency checking only applies to LifeTimeInCache beans whose data is read from another transaction.

Usage scenario

For the Access Intents that are for *repeatable read* (RR), this means the product checks that the data is consistent with that in the data store and ensures that no one updates it after the checking. For the Access Intents that are for *read committed* (RC), this means the product checks that the data is consistent at the point of checking, but it does **not** guarantee that the data does not change after the checking. This makes the behavior of the LifeTimeInCache bean the same as non-LifeTimeInCache beans.

You have three options for setting consistency checking, as shown in the following scenarios concerning the calculation of interest in ″Ann's″ bank account. In each case, the data store is shared by this EJB CMP application (to calculate the interest) and other applications, such as EJB BMP, JDBC, or legacy applications. Also in each case, the EJB Account is configured as a “long-lifetime” bean.

NONE

- The server is started.
- User 1 in Transaction 1 calls Account.findByPrimaryKey(″10001″), account data for Ann is read from the database, with a balance of $100.
- Ann's record is cached by the persistence manager (PM) on the server.
- User 2 writes a JDBC call and changes the balance to $120.
- User 3 in Transaction 2 calls Account.findByPrimaryKey() for account ″10001″, Ann's data is read from cache, with a balance of $100.
- Calculate Ann's interest, but the result might not be correct because of the data integrity issue.

Read-read checking AT_TRAN_BEGIN

- The server is started.
- User 1 in Transaction 1 calls Account.findByPrimaryKey(″10001″), account data for Ann is read from the database, with a balance of $100.
- Ann's record is cached by the persistence manager (PM) on the server.
- User 2 writes a JDBC call and changes the balance to $120.
- User 3 in Transaction 2 calls Account.findByPrimaryKey() for account ″10001″, Ann's data is read from cache, with a balance of $100.
- PM performs read-read check on Ann's account and finds that the balance of 100 is changed. It issues a database query to retrieve balance of $120, and Ann's data in the cache is refreshed.
- Calculate Ann's interest, proceed with the transaction because data integrity is protected.

Read-read checking AT_TRAN_END

- The server is started.
- User 1 in Transaction 1 calls Account.findByPrimaryKey(″10001″), account data for Ann is read from the database, with a balance of $100.
- Ann's record is cached by the persistence manager (PM) on the server.
- User 2 writes a JDBC call and changes the balance to $120.
- User 3 in Transaction 2 calls Account.findByPrimaryKey() for account ″10001″, Ann's data is read from database, with balance of $100.
- Calculate Ann's interest.
- During end of transaction 2, PM performs read-read check on Ann's account and finds that the balance of 100 is changed.
- PM rolls back the transaction and invalidates the cache. The transaction fails and again data integrity is protected.

Access intent service

Access intent is a WebSphere Application Server runtime service that enables you to more precisely manage an application's persistence.

The access intent service defines a set of declarative annotations used by the Enterprise JavaBeans (EJB) container and its agents to make performance optimizations for entity bean access. These annotations are organized into sets called *access intent policies*.

Access intent policies contain a set of annotations considered as hints by the EJB container and its agents. Most access intent policies are hints representing high-level abstractions that can be mapped to a specific back end resource manager. It is the responsibility of the EJB persistence machinery to ensure the necessary concurrency control, connection, and cache management when carrying out the persistence details. The EJB persistence manager can use access intent hints to make better performance decisions when carrying out its assigned task. A smaller number of access intents are hints to the EJB container, influencing the management of EJB collections.

Generally you configure *bean level* access intent for your applications. You can also apply access intent policies to beans within the scope of *application profiles*. Consequently, you can configure beans with multiple and opposing access intent policies. The application profiling documentation explains in more detail how to configure an application to apply a particular access intent policy to a bean for one request, then apply another access intent policy to the same bean for a different request.

Support for applying access intent policies at the method level is deprecated in WebSphere Application Server Version 6.0. In this practice of configuring access intent, you apply a policy to methods within the scope of an EJB module so that the policy becomes the default access intent for all requests upon those methods.

Access intent with BMP entity beans

Access intent's declarative functionality provides great power to you as a CMP entity bean developer. You can provide hints on how WebSphere Application Server is to manage the details of persistence without having to explicitly manage any of the persistence logic from within the application.

There are situations, however, in which you might need to develop BMP entity beans. Because the only meaningful difference between BMP and CMP components is who provides the persistence logic, BMP entity beans should be able to leverage access intent hints just as WebSphere Application Server does on behalf of CMP entity beans. BMP entity beans that use the access intent service participate in application profiling; that is, the value of the access intent attributes can differ from request to request, allowing the BMP entity bean to seamlessly modify its persistence strategy.

You can apply access intent policies to BMP entity bean methods as well as CMP entity bean methods. Because access intent hints are not contractual in nature, there is no obligation for a BMP entity bean to exploit them. BMP entity beans are expected to use only those access intent attributes that are important to that particular bean.

The current access intent policy is bound into the java:comp namespace for a particular BMP entity bean. That policy is current only for the duration of the method call during which the access intent policy was retrieved. In a typical scenario, you would cache the access type during invocation of the ejbLoad() method so that appropriate actions can be taken during invocation of the ejbStore() method.

Access intent design considerations

Use the access intent service to solve clear performance problems. Identify usage patterns that lead to poor application performance and apply appropriate access intent policies.

best-practices: Refrain from over-tuning an application. You can introduce errors by incorrectly using the access intent service. For example, misuse of the wsPessimisticUpdate-NoCollision policy can result in lost updates; inappropriately setting the collection increment value can introduce performance issues; and problem determination is more difficult when an application is confusingly configured with multiple access intent policies.

Note: Clarity and simplicity should be your guiding principles when using the access intent service. This is even more important when applying access intent polices within the scope of application profiles.

Even though access intent policies can be configured on any method of an entity bean, some attributes of a policy can only be leveraged by the runtime environment under certain conditions. For example,

concurrency and access intent are only used for CMP entity beans when the ejbLoad() method is driven to open a connection to, and read data from, a given resource; that data is cached and used to drive the proper queries during invocation of the ejbStore() method. Read-ahead hints are only used during the execution of a finder for a bean. Finally, the collection increment and resource manager prefetch increment are only used on multi-object finders. Configuring policies on methods that will not use the policy is not an error (only certain attributes of any policy are used, even when the policy is appropriately applied to a method). However, configuring policies unnecessarily throughout an application obscures the design of the application and complicates the maintenance of the application.

Applying access intent policies to methods

You apply an access intent policy to a method, or set of methods, in an application's entity beans through the assembly tool.

Note: Method-level access intent is deprecated in Version 6.0.

1. Start the Application Server Toolkit.
2. **Optional:** Open the J2EE perspective to work with J2EE projects. Click **Window > Open Perspective > Other > J2EE**.
3. **Optional:** Open the Project Explorer view. Click **Window > Show View > Project Explorer**. Another helpful view is the Navigator view (**Window > Show View > Navigator**).
4. Create a new application EAR file or edit an existing one.

   For example, to change attributes of an existing application, use the import wizard to import an EAR file. To start the import wizard:

   a. Select **File > Import > EAR file > Next**
   b. Select the EAR file.
   c. Create a WebSphere Application Server v6.0 type of Server Runtime. Select **New** to open the New Server Runtime Wizard and follow the instructions.
   d. In the *Target server* field, select *WebSphere Application Server v6.0* type of Server Runtime.
   e. Select **Finish**
5. In the Project Explorer view of the J2EE perspective, right-click the **Deployment Descriptor: EJB Module Name** under the EJB module for the bean instance, then select **Open With > Deployment Descriptor Editor**. A property dialog notebook for the EJB project is displayed in the property pane.
6. Select the **Access** tab.
7. On the right side of the **Access Intent for Entities 2.x (Method Level)** panel, select **Add**. The **Add Access Intent** panel displays.
8. Specify the **Name** for your new intent policy.
9. Select the **Access intent name** from the drop-down list.
10. Enter a **Description** to help you remember what this policy does.
11. **Optional:** Select **Read Ahead Hint**. A single access intent read ahead hint might not refer to the same bean type in more than one relationship. For example, if a **Department** enterprise bean has a relationship *employees* with the **Employee** enterprise bean, and also has a relationship *manager* with the **Employee** enterprise bean, then a read ahead hint cannot specify both *employees* and *manager*.
12. Click **Next**. The next **Add Access Intent** panel displays, with optional attributes.
13. **Optional:** Decide whether or not to overwrite these optional access intent attributes. Click on those you want to change.
14. Click **Next**. The next **Add Access Intent** panel, with a list of Enterprise Beans, displays.
15. Select one or more Enterprise Beans from the list.

    Note: If you selected **Read Ahead Hint** in an earlier step, you can only select **ONE** bean at this step.
16. Click **Next**. The next **Add Access Intent** panel, with a list of methods, displays.

17. Select the methods you want to use.
18. If you *DID NOT* select **Read Ahead Hint** in an earlier step, click **Finish**. If you *DID* select the Read Ahead Hint option, you can click **Next** to specify your Read Ahead Hint for the specified bean. The next **Add Access Intent** panel, with a list of EJB preload paths, displays.
19. Edit the EJB preload path by selecting relationship roles from the **Relationship roles:** window.
20. Click **Finish**. A new entry is created in the **Access Intent for Entities 2.x (Method Level)** panel

Using the AccessIntent API

This task describes how to programmatically retrieve and call the AccessIntent API during the execution of BMP entity bean methods.

1. Look up the current access intent in the namespace. For example:

```
InitialContext ic = new InitialContext();
AccessIntent ai = ic.lookup("java:comp/websphere/AppProfile/AccessIntent");
```

2. Call the necessary get() methods. For example:

```
int concurrency = ai.getConcurrencyControl();
int accessType = ai.getAccessType();
if ( (concurrency == AccessIntent.CONCURRENCY_CONTROL_PESSIMISTIC)
   && (accessType == AccessIntent.ACCESS_TYPE_UPDATE) ) {
     int exclusive = ai.getPessimisticUpdateLockHint();
     // . . .
}
// . . .
```

Note: The access intent object reference retrieved from the java:comp lookup is current for the duration of the method in which the reference was looked up. Depending on how you configured the application profile, subsequent calls of the same method might not retrieve the same access intent reference. You can only look up the object reference during the call of a BMP entity bean's method; the reference does not exist during a request on a CMP entity bean. Therefore, access intent object references should not be cached beyond, or used outside of, the scope of the execution of any given BMP method.

AccessIntent interface

The AccessIntent interface is available to BMP entity beans.

The following JNDI lookup allows BMP entity beans to access the AccessIntent interface:

```
java:comp/websphere/AppProfile/AccessIntent
```

AccessIntent interface

```
package com.ibm.websphere.appprofile.accessintent;

/**
* This interface defines the essential access intents
* available at run time.
*/
public interface AccessIntent {

/**
* Returns the concurrency control intent, which indicates
* the application prefers either pessimistic or optimistic
* concurrency control when accessing the current component
* in the context of the current transaction.
*/
public int getConcurrencyControl();
public final int CONCURRENCY_CONTROL_PESSIMISTIC = 1;
public final int CONCURRENCY_CONTROL_OPTIMISTIC = 2;

/**
* Returns access type intent, which indicates the application
```

```
* intends either update or read access of the current component
* in the context of the current transaction.
*/
public int getAccessType();
public final int ACCESS_TYPE_UPDATE= 1;
public final int ACCESS_TYPE_READ = 2;

/**
* Returns an integer value that indicates that the run time should
* assume that there will be no collision on retrieved rows.
*/
public int getPessimisticUpdateLockHint();
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NOCOLLISION = 1;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_WEAKEST_LOCK_AT_LOAD = 2;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NONE = 3;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_EXCLUSIVE = 4;

/*
* Returns an integer value that indicates that the run time should
* assume that there will be collisions on retrieved rows.
*/
public int getPessimisticUpdateLockHint();
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NOCOLLISION = 1;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_WEAKEST_LOCK_AT_LOAD = 2;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NONE = 3;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_EXCLUSIVE = 4;


/**
* Returns the collection access intent, which indicates the
* application intends to access the objects returned by the
* currently executing finder in either serial or random fashion.
*/
public int getCollectionAccess();
public final int COLLECTION_ACCESS_RANDOM = 1;
public final int COLLECTION_ACCESS_SERIAL = 2;

/**
* Returns the collection scope, which indicates the maximum
* lifespan of a lazy collection.
*/
public int getCollectionScope();
public final int COLLECTION_SCOPE_TRANSACTION = 1;
public final int COLLECTION_SCOPE_ACTIVITYSESSION = 2;
public final int COLLECTION_SCOPE_TIMEOUT = 3;

/**
* Returns the timeout value in seconds when collectionScope is Timeout.
*/
public int getCollectionTimeout();

/**
* Returns the number of elements the application requests be contained
* in each segment of the element collection returned by the currently
* executing finder.
*/
public int getCollectionIncrement();

/**
* Returns the ReadAheadHint requested by the application for the currently
* executing finder.
*/
public ReadAheadHint getReadAheadHint();

/**
* Returns the number of elements the application requests be contained in
* each segment of a query made on a database.
```

```
*/
public int getResourceManagerPreFetchIncrement();

}
```

Access intent exceptions

The exceptions thrown in response to the application of access intent policies are listed.

com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException

If the method that drives the ejbLoad() method is configured to be read-only but updates are then made within the transaction that loaded the bean's state, an exception is thrown during invocation of the ejbStore() method, and the transaction is rolled back. Likewise, the ejbRemove() method cannot succeed in a transaction that is set as read-only. If an update hint is applied to methods of entity beans with bean-managed persistence, the same behavior and exception results. The forwarded exception object contains the message string `PMGR1103E: update instance level read only bean` *beanName*

This exception is also thrown if the applied access intent policy cannot be honored because a finder, ejbSelect, or container-managed relationship (CMR) accessor method returns an inherently read-only result. The forwarded exception object contains the message string `PMGR1001: No such DataAccessSpec -` *methodName*

The most common occurrence of this error is when a custom finder that contains a read-only EJB Query Language (EJB QL) statement is called with an applied access intent of `wsPessimisticUpdate` or `wsPessimisticUpdate-Exclusive`. These policies require the use of a USE AND KEEP UPDATE LOCKS clause on the SQL SELECT statement to be executed, but a read-only query cannot support USE AND KEEP UPDATE LOCKS. Other examples of read-only queries include joins; the use of ORDER BY, GROUP BY, and DISTINCT keywords.

To eliminate the exception, edit the EJB query so that it does not return an inherently read-only result or change the access intent policy being applied.

- If an update access is required, change the applied access intent setting to `wsPessimisticUpdate-WeakestLockAtLoad` or `wsOptimisticUpdate`.
- If update access is not truly required, use `wsPessimisticRead` or `wsOptimisticRead`.
- If connection sharing between entity beans is required, use `wsPessimisticUpdate-WeakestLockAtLoad` or `wsPessimisticRead`.

com.ibm.websphere.ejb.container.CollectionCannotBeFurtherAccessed

If a lazy collection is driven after it is no longer in scope, and beyond what has already been locally buffered, a CollectionCannotBeFurtherAccessed exception is thrown.

com.ibm.ws.exception.RuntimeWarning

If an application is configured incorrectly, a run-time warning exception is thrown as the application starts; startup is ended. You can validate an application's configuration by choosing the verify function. Some examples of misconfiguration include:

- A method configured with two different access intent policies
- A method configured with an undefined access intent policy

Access intent best practices

When applying access intent policies to Enterprise JavaBeans (EJB) methods, consider the following issues.

- **Start by configuring the default access intent policy for an entity.** After your application is built and running, you can more finely tune certain access paths in your application using application profiling or method-level access intent.
- **Don't mix access types.** Avoid using both pessimistic and optimistic policies in the same transaction. For most databases, pessimistic and optimistic policies use different isolation levels. This can result in multiple database connections, which prevents you from taking advantage of the performance benefits possible through connection sharing.

- **Take care when applying `wsPessimisticUpdate-NoCollision`.** This policy does not ensure data integrity. No database locks are held, so concurrent transactions can overwrite each other's updates. Use this policy only if you can be sure that only one transaction will attempt to update persistent store at any given time.

Frequently asked questions: Access intent

The following frequently asked questions involving access intent are answered.

I have not applied any access intent policies at all. My application runs just fine with a DB2 database, but it fails with an Oracle database with the following message: *com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException: PMGR1001E: No such DataAccessSpec :FindAllCustomers. The backend datastore does not support the SQLStatement needed by this AccessIntent: (pessimistic update-weakestLockAtLoad)(collections: transaction/25) (resource manager prefetch: 0) (AccessIntentImpl@d23690a)*. Why?

If you have not configured access intent, all of your data is accessed under the default access intent policy (`wsPessimisticUpdate-WeakestLockAtLoad`). On DB2 the weakest lock is share. On Oracle databases, however, the weakest lock is update; this means that the SQL query must contain a FOR UPDATE clause. To avoid this problem, try to apply an access intent policy that supports optimistic concurrency.

I am calling a finder method and I get an InconsistentAccessIntentException at run time. Why?

This can occur when you use method-level access intent policies to apply more control over how a bean instance is loaded. This exception indicates that the entity bean was previously loaded in the same transaction. This could happen if you called a multifinder method that returned the bean instance with access intent policy X applied; you are now trying to load the second bean again by calling its findByPrimaryKey method with access intent Y applied. Both methods must have the same access intent policy applied.

Likewise, if the entity was loaded once in the transaction using an access intent policy configured on a finder, you might have called a container-managed relationship (CMR) accessor method that returned the entity bean configured to load using that entity's default access intent.

To avoid this problem, ensure that your code does not load the same bean instance twice within the same transaction with different access intent policies applied. Avoid the use of method-level access intent unless absolutely necessary.

I have two beans in a container-managed relationship. I call findByPrimaryKey() on the first bean and then call getBean2( ), a CMR accessor method, on the returned instance. At that point, I get an InconsistentAccessIntentException. Why?

You are probably using read-ahead. When you loaded the first bean, you caused the second bean to be loaded under the access intent policy applied to the finder method for the first bean. However, you have configured your CMR accessor method from the first bean to the second with a different access intent policy. CMR accessor methods are really finder methods in disguise; the run-time environment behaves as if you were trying to change the access intent for an instance you have already read from persistent store.

To avoid this problem, beans configured in a read-ahead hint are all driven to load with the same access intent policy as the bean to which the read-ahead hint is applied.

I have a bean with a one-to-many relationship to a second bean. The first bean has a pessimistic-update intent policy applied. When I try to add an instance of the second bean to the first bean's collection, I get an UpdateCannotProceedWithIntegrityException. Why?

The second bean probably has a read intent policy applied. When you add the second bean to the first bean's collection, you are not updating the first bean's state, you are implicitly modifying the second bean's state. (The second bean contains a foreign key to the first bean, which is modified.)

To avoid this problem, ensure that both ends of the relationship have an update intent policy applied if you expect to change the relationship at run time.

Managing EJB containers

Each application server can have a single EJB container; one is created automatically for you when the application server is created. The following steps are to be performed only as needed to improve performance after the EJB application has been deployed.

1. Adjust EJB container settings.
2. Adjust EJB cache settings.

If adjustments do not improve performance, consider adjusting access intent policies for entity beans, reassembling the module, and redeploying the module in the application.

EJB container settings

Use this page to configure and manage the EJB container of this application server.

To view this administrative console page, click **Servers > Application Servers >** *serverName* **> EJB Container Settings** > **EJB container**.

Passivation directory

Specifies the directory into which the container saves the persistent state of passivated stateful session beans. This directory must already exist. It is not automatically created.

Stateful session beans with an activation policy of TRANSACTION are passivated at the end of the transaction in which they are enlisted, and stateful session beans with an activation policy of ONCE (default) are passivated when the number of active bean instances becomes greater than the cache size specified in the container configuration. When a stateful bean is passivated, the container serializes the bean instance to a file in the passivation directory and discards the instance from the bean cache. If, at a later time, a request arrives for the passivated bean instance, the container retrieves it from the passivation directory, deserializes it, returns it to the cache, and dispatches the request to it. If any step fails (for example, if the bean instance is no longer in the passivation directory), the method invocation fails.

Inactive pool cleanup interval

Specifies the interval at which the container examines the pools of available bean instances to determine if some instances can be deleted to reduce memory usage.

Data type	Integer
Units	Milliseconds
Range	0 to 2 147 483 647

Default data source JNDI name

Specifies the JNDI name of a data source to use if no data source is specified during application deployment. This setting is not applicable for EJB 2.x-compliant CMP beans.

Servlets and enterprise beans use *data sources* to obtain these connections. When configuring a container, you can specify a default data source for the container. This data source becomes the default data source used by any entity beans installed in the container that use container-managed persistence (CMP).

The default data source for a container is secure. When specifying it, you must provide a user ID and password for accessing the data source.

Specifying a default data source is optional if each CMP entity bean in the container has a data source specified in its configuration. If a default data source is not specified and a CMP entity bean is installed in the container without specifying a data source for that bean, applications cannot use that CMP entity bean.

Enable stateful session bean failover using memory-to-memory replication

Specifies that failover is enabled for *all* stateful session beans installed in this EJB container.

This checkbox is disabled until you define a replication domain. This selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a panel where you can create one. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the EJB container.

Data type	Checkbox
Default	Unselected
Range	Selected or unselected.

Initial state

Specifies the execution state requested when the server first starts.

Data type	String
Default	Started
Range	Valid values are `Started` and `Stopped`

EJB container system properties

In addition to the settings accessible from the administrative console, you can set the following system property by command-line scripting.

com.ibm.websphere.ejbcontainer.poolSize

Specifies the size of the pool for the specified bean type. This property applies to stateless, message-driven and entity beans. If you do not specify a default value, the container defaults of 50 and 500 are used.

Set the pool size for a given entity bean as follows:

```
beantype=min,[H]max [:beantype=min,max...]
```

beantype is the J2EE name of the bean, formed by concatenating the application name, the **#** character, the module name, the **#** character, and the name of the bean (that is, the string assigned to the `<ejb-name>` field in the bean's deployment descriptor). *min* and *max* are the minimum and maximum pool sizes, respectively, for that bean type. Do not specify the square brackets shown in the previous prototype; they denote optional additional bean types that you can specify after the first. Each bean-type specification is delimited by a colon (:).

Use an asterisk (*) as the value of *beantype* to indicate that all bean types are to use those values unless overridden by an exact bean-type specification somewhere else in the string, as follows:

```
*=30,100
```

To specify that a default value be used, omit either *min* or *max* but retain the comma (,) between the two values, as follows (split for publication):

```
SMApp#PerfModule#TunerBean=54,
   :SMApp#SMModule#TypeBean=100,200
```

You can specify the bean types in any order within the string.

com.ibm.websphere.ejbcontainer.allowEarlyInsert

Note: This property is applicable to CMP 1.1 beans only.

By default, the EJB Container creates the entity bean representation in the database only after the method ejbPostCreate(...) is called. However, some applications may rely on method ejbCreate(...) to have created the entity bean in the database. For such a requirement, setting the JVM property *com.ibm.websphere.ejbcontainer.allowEarlyInsert* to **true** overrides the default behavior.

Changing enterprise bean types to initialize at application start time using the Application Server Toolkit

EJB types can be forced to initialize at application start time by setting a flag within the bean's deployment descriptor. If this flag is set to **true**, then the bean is initialized at application start time.

However, by default, the WebSphere Application Server's Enterprise JavaBeans (EJB) Container delays the initialization (loading and processing) of most EJB types until they are needed during runtime. This delay helps to speed up the application start time.

1. Start the Application Server Toolkit.
2. Select **EJB Deployment Descriptor**.
3. In the property pane, select the **WebSphere Extensions** tab.
4. Check the box labeled **Start EJB at Application Start**.
5. Select **OK**.

Changing enterprise bean types to initialize at application start time using the administrative console

All EJB types within a server can be forced to initialize at application start time by setting a system property within the administrative console. If the value of this property is set to **true**, then all beans within the server are initialized at each application's start time.

However, by default, the WebSphere Application Server's Enterprise JavaBeans (EJB) Container delays the initialization (loading of classes and processing of deployment descriptor metadata) of most EJB types until they are needed during run time. This delay helps to speed up the application start time.

1. Open the administrative console.
2. Select **Servers**.
3. Select **Application Servers**.
4. Select the server you want to configure.
5. In the Server Infrastructure area, select **Java and Process Management**.
6. In the Server Infrastructure area, select **Process Definition**.
7. In the Additional Properties area, select **Java Virtual Machine**.
8. In the Additional Properties area, select **Custom Properties**.
9. Select the **New** box.
10. In the **Name** entry field, type *com.ibm.websphere.ejbcontainer.initializeEJBsAtStartup*.
11. In the **Value** entry field, type *true*. Entering *true* causes all Enterprise JavaBeans to initialize when your application starts. Entering *false* causes initialization of all beans to be delayed.

    Note: Setting com.ibm.websphere.ejbcontainer.initializeEJBsAtStartup to either true or false takes precedence over any *Start EJB at Application Start* settings made on individual EJB types (see "Changing enterprise bean types to initialize at application start time using the Application Server Toolkit").
12. Select **OK**.

EJB cache settings

Use this page to configure and manage the cache for a specific EJB container. To avoid errors from attempting to overload the cache, determine the cache absolute limit. Multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then, add the number of active session bean instances. This value is the limit that the cache will hold. You can use the Tivoli Performance Viewer to view bean performance information.

To view this administrative console page, click **Servers > Application Servers >** *serverName* **> EJB Container Settings > EJB cache settings**.

Cleanup interval

Specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size.

The cache manager tries to maintain some unallocated entries that can be allocated quickly as needed. A background thread attempts to free some entries while maintaining some unallocated entries. If the thread runs while the application server is idle, when the application server needs to allocate new cache entries, it does not pay the performance cost of removing entries from the cache. In general, increase this parameter as the cache size increases.

Data type	Integer
Units	Milliseconds
Range	0 to 2 147 483 647
Default	3000

Cache size

Specifies the number of buckets in the active instance list within the EJB container.

A bucket can contain more than one active enterprise bean instance, but performance is maximized if each bucket in the table has a minimum number of instances assigned to it. When the number of active instances within the container exceeds the number of buckets, that is, the cache size, the container periodically attempts to reduce the number of active instances in the table by passivating some of the active instances. For the best balance of performance and memory, set this value to the maximum number of active instances expected during a typical workload.

Data type	Integer
Units	Buckets in the hash table
Range	Greater than 0. The container selects the next largest prime number equal to or greater than the specified value.
Default	2053

Container interoperability

Container interoperability describes the ability of WebSphere Application Server clients and servers at different versions to successfully negotiate differences in native Enterprise JavaBeans (EJB) finder methods support and Java 2 Platform, Enterprise Edition (J2EE) compliance.

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

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

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

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

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

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

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

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

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

Replication of the Http Session and Handles

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

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

Top Down Deployment Mapping

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

EJB Container tuning

If you use applications that affect the size of the EJB Container Cache, it is possible that the performance of your applications can be impacted by an incorrect size setting. Monitoring Tivoli Performance Viewer (TPV) is a great way to diagnose if the EJB Container Cache size setting is tuned correctly for your application.

If the application has filled the cache causing evictions to occur, TPV will show a very high rate of ejbStores() being called and probably a lower than expected CPU utilization on the application server machine.

All applications using enterprise beans should have this setting adjusted from the default if the following formula works out to more than 2000.

```
EJB_Cache_Size = (Largest number of Option B or C Entity Beans enlisted in a
transaction * maximum number of concurrent transactions) +
(Largest number of unique Option A Entity Beans expected to be accessed during
typical application workload) +
(Number of stateful Session Beans active during typical workload) +
(Number of stateless SessionBean types used during typical workload)

Where:
Option B and C Entity Beans are only held in the EJB cache during the lifetime
of the transaction they are enlisted in. Therefore, the first term in the formula
computes the average EJB cache requirements for these types of beans.

Option A Entity Beans are held in the EJB cache indefinitely, and are only removed
 from the cache if there start to become more beans in the cache than the cache
size has been set to.

Stateful Session Beans are held in the EJB cache until they are removed by the
application, or their session timeout value is reached.

Only a single stateless Session Bean instance for each EJB type is held in the
cache during the time any methods are being executed on that stateless Session
Bean. If two or more methods are being executed simultaneously on the same
stateless Session Bean type, each method executes on its own bean instance, but
only one cache location is used for all of these instances.
```

This calculates the upper bound on the maximum possible number of enterprise beans active at one time inside the application server. Because the EJB Containers cache is built to contain all these beans for performance optimizations, best performance can be achieved by setting this cache size to be larger than the number resulting from the calculation above.

```
<tuning parameter>
This setting can be found under Servers > Application Servers > serverName >
         EJB Container > EJB Cache Settings
```

Also while adjusting the EJB Cache Size, the EJB Container management thread parameter can be tuned to meet the needs of the application. The management thread is controlled through the Clean Up Interval setting. This setting controls how frequently a daemon thread inside of WebSphere Application Server wakes up and attempts to remove bean instances from the cache that have not been used recently, attempting to keep the number of bean instances at or below the cache size. This allows the EJB container to place and look up items in the cache as quickly as possible. It normally is best to leave this interval set to the default, however, in some cases, it may be worthwhile to see if there is a benefit to reducing this interval.

EJB Container Pool Size

If the application is using the majority of the instances in the pool, TPV indicates this. When this occurs, then the size of those bean pools that are being exhausted should be increased. This can be done by adding the following parameter in the JVM's custom properties tag .

```
-Dcom.ibm.websphere.ejbcontainer.poolSize=<application_name>#<module_name>#
<enterprisebean_name>=<minSize>,<maxSize>

where:
 <application_name> is the J2EE application name as defined in the application
archive (.ear) file deployment descriptor, for the bean whose pool size is being
set

<module_name> is the .jar file name of the EJB module, for the bean whose pool size
 is being set,

<bean_name> is the J2EE Enterprise Bean name as defined in the EJB module
deployment descriptor, for the bean whose pool size is being set

<minSize> is the number of bean instances the container maintains in the pool,
 irrespective of how long the beans have been in the pool (beans greater than this
 number are cleared from the pool over time to optimize memory usage)

<maxSize> is the number of bean instances in the pool where no more bean instances
 are placed in the pool after they are used (that is, once the pool is at this
 size, any additional beans are discarded rather than added into the pool --
this ensures the number of beans in the pool has an upper limit so memory usage
does not grow in an unbounded fashion).

To keep the number of instances in the pool at a fixed size, minSize and maxSize
can be set to the same number. Note that there is a separate instance pool for
every EJB type running in the application server, and that every pool starts out
with no instances in it - that is, the number of instances grows as beans are
used and then placed in the pool. When a bean instance is needed by the container
and no beans are available in the pool, the container creates a new bean
instance, uses it, then places that instance in the pool (unless there are
already maxSize instances in the pool).

For example, the statement
-Dcom.ibm.websphere.ejbcontainer.poolSize=ivtApp#ivtEJB.jar#ivtEJBObject=125,1327

would set a minSize of 125 and a maxSize of 1327 on the bean named "ivtEJBObject"
within the ivtEJB.jar file, in the application "ivtApp".
```

Where ivtApp is replaced by the actual application name, ivtEJB.jar is replaced by the jar containing the bean that needs to have its pool size increased, and ivtEJBObject is the bean name of the enterprise bean whose pool size should be increased. The 125,1327 is the minimum and maximum number of beans that will be held in the pool. These should be set so no more evictions occur from the pool and in most cases should be set equal if memory is plentiful because no growth and shrinkage of the pool will occur.

EJB Container Primary Key Mutation

Application developers and administrators should have a good idea of how their application handles the creation of primary key objects for use by container-managed persistence (CMP) beans and bean-managed persistence (BMP) beans inside of WebSphere Application Server. The IBM EJB Container uses the primary key of an Entity bean as an identifier inside of many internal data structures to optimize performance. However, the EJB Container must copy these primary key objects upon the first access to the bean to ensure that the objects stored in the internal caches are separate from the ones used in an application, in case the application changes or mutates the primary key, to keep the internal structures consistent.

If the application does not mutate any of the primary keys used to create and access entity beans after they are created, then a special flag can be used that allows the EJB Container to skip the copy of the primary key object, thus saving CPU cycles and increasing performance. This mechanism can be enabled *at your own risk* by adding the following –D property to the JVM custom property field.

```
<tuning parameter>
-Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation=true
```

The performance benefit of this optimization depends on the application. If the application uses primitive types for enterprise beans' primary keys there will be no gain because these objects are already immutable and the copy mechanism takes this into account. If, however, the application uses many complex primary keys (that is, And object for a primary key or multiple fields) then this parameter can yield significant improvements.

Persistence Manager Deferred Insert on EJB Create

The IBM Persistence manager is used by the EJB Container to persist data to the database from CMP entity beans. When creating entity beans by calling the ejbCreate() method, by default the Persistence manager immediately inserts the empty row with only the primary key in the database. In most cases applications, after creating the bean, modify fields in the bean created or in other beans inside of the same transaction. If the user wishes to postpone the insert into the database until the end of the transaction, so that it will eliminate one trip to the database, they may set this –D flag inside of the JVM custom properties field. The data will still be inserted into the database and consistency will be maintained.

```
<tuning parameter>
-Dcom.ibm.ws.pm.deferredcreate=true
```

The performance benefit of this optimization depends on the application. If the EJB applications transactions are very insert intensive the application could benefit largely from this optimization. If the application performs very few inserts then the benefit of this optimization will be much less.

Persistence Manager Database Batch Update on EJB Update

When an EJB application accesses multiple CMP beans inside of a single transaction, depending on the operations performed on the beans (updates, inserts, reads), the number of operations issued to the database will correspond directly to the operations performed on the CMP beans. If the database system you are using supports batching of update statements you can enable this flag and gain a performance boost on all interactions with the database that involve more than two updates in a single transaction. This flag will let the persistence manager add all the update statements into one single batch statement which will then be issued to the database. This saves round trips to the database, thus increasing performance. If the user knows their application exhibits the behavior of updating multiple CMP beans in a single transaction and the database supports batch updates they may set this –D flag inside of the JVM custom properties field.

```
<tuning parameter>
-Dcom.ibm.ws.pm.batch=true
```

The performance benefit of this optimization depends on the application. If the application never or infrequently updates CMP beans or only updates a single bean per transaction there will be no performance gain. If the application updates multiple beans per transaction then this parameter will benefit your applications performance.

The following table lists which backend databases support batch update.

Table 5.

Database	Supports Batch update	Supports Batch update with Optimistic Concurrency Control
DB2	yes	no
Oracle	yes	no
DB2 Universal Driver	yes	yes
Informix	yes	yes
SQLServer	yes	yes
Cloudscape	yes	yes

Note: Batch update with OCC cannot be performed for databases that do not support it, even if specified by the access intent.

Persistence Manager cache Tuning

Persistence Manager has two different types of caching mechanisms available: *legacy cache* and *two-level cache*. Normally two-level cache performs better than legacy cache because of optimizations in this mode. The default is legacy cache, although two-level cache is recommended. Set this configuration through the system property

```
com.ibm.ws.pm.useLegacyCache=false
```

Persistence Manager Partial Updates Tuning

The partial updates feature enhances the performance of applications with enterprise beans in certain scenarios. Persistence Manager has two different types of caching mechanisms available, legacy cache and two-level cache. Normally, two-level cache performs better than legacy cache because of the optimizations in this mode. In certain applications where you need to perform both batch updates and partial updates, you must configure the following system properties to gain the benefits of both.

```
'com.ibm.ws.pm.grouppartialupdate=true' and 'com.ibm.ws.pm.batch=true'
```

Deploying EJB modules

When you deploy an EJB module, you install that module on a server that has been configured to support deployed modules.

Assemble one or more EJB modules, assemble one or more Web modules, and assemble them into a J2EE application.

1. Prepare the deployment environment.
2. Update the configuration for each EJB module as needed for the deployment environment. See the AST information center for more information about modifying deployment descriptors.
3. Deploy the application.

If you specify that EJB deploy 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.

Either remove the source files or include all dependent classes and resource files on the class path. If there are source files in the input JAR or EAR file, the EJB deployment tools runs a rebuild before generating the deployment code.

If the module deploys successfully, test and debug the module.

Troubleshooting tips for EJBDEPLOY relationships

This article provides troubleshooting information for EJBDEPLOY problems.

The converter that is defined for the primary key is not invoked on its foreign key value

The mapping for primary key fields to database columns may use a converter to transform the key values. If a container-managed persistence (CMP) bean uses a converter to map its primary key, and that bean has a relationship where the bean at the other end holds a foreign key, the mapping for the foreign key will not use the converter.

The following errors might occur, indicating that the converter defined for the primary key is not invoked on its foreign key value. During the run of the ejbDeploy command , you receive the following message:

```
No type mapping defined for Java datatype1 to Database datatype2
```

During run time, the application does not find the CMP bean at the other end of the relationship.

To work around this limitation, define your own foreign key in the database table, and create a mapping that uses the same converter as defined for the primary key on the enterprise beans at the other end of its relationship.

EJB module settings

Use this page to configure and manage a specific deployed EJB module.

Note: You cannot start or stop an individual EJB module for modification. You must start or stop the appropriate application entirely.

To view this administrative console page, click **Applications > Enterprise Applications >** *applicationName* **> Manage Modules >** *moduleName*.

URI

Specifies location of the module relative to the root of the application EAR file. The URI must match the URI of a ModuleRef URI in the deployment descriptor of the deployed application (EAR).

Alternate deployment descriptor

Specifies an alternate deployment descriptor for the module as defined in the application deployment descriptor according to the J2EE specification.

Starting weight

Specifies the order in which modules are started when the server starts. The module with the lowest starting weight is started first.

Data type	Integer
Default	5000
Range	Greater than 0

Chapter 11. Client applications

Using application clients

An application client module is a Java Archive (JAR) file that contains a client for accessing a Java application.

Complete the following steps for developing different types of application clients.

1. Decide on a type of application client.
2. Develop the application client code.
   a. Develop ActiveX application client code.
   b. Develop J2EE application client code.
   c. Develop pluggable application client code.
   d. Develop thin application client code.
3. Assemble the application client using the Application Server Toolkit.
4. Deploy the application client.
   Deploy the application client on Windows systems.
5. Run the application client.

View the Application Clients Samples Gallery for more information. To access these samples, install Application Clients, and retrieve the samples from your local file system as the following command indicates:

`<app_server_root>/samples/index.html`

Application Client for WebSphere Application Server

In a traditional client-server environment, the client requests a service and the server fulfills the request. Multiple clients use a single server. Clients can also access several different servers. This model persists for Java clients except that now these requests use a client runtime environment.

WebSphere Application Server Version 6.1 supports the pluggable client.

In this model, the client application requires a servlet to communicate with the enterprise bean, and the servlet must reside on the same machine as the WebSphere Application Server.

The Application Client for WebSphere Application Server Version 6 (Application Client) consists of the following client applications:

- J2EE application client application (Uses services provided by the J2EE Client Container)
- Thin application client application (Does not use services provided by the J2EE Client Container)
- Applet application client application
- ActiveX to EJB Bridge application client application (Windows only)

The Application Client is packaged with the following components:

- Java Runtime Environment (JRE) (or an optional full Software Development Kit) that IBM provides.
- WebSphere Application Server run time for J2EE application client applications or Thin application client applications
- Windows An ActiveX to EJB Bridge run time for ActiveX to EJB Bridge application client applications (Windows only)
- Windows IBM plug-in for Java platforms for Applet client applications (Windows only)

Note: The Pluggable application client is a kind of Thin application client. However, the Pluggable application client uses a Sun JRE and Software Development Kit instead of the JRE and Software Development Kit that IBM provides.

The *ActiveX application client* model, uses the Java Native Interface (JNI) architecture to programmatically access the Java virtual machine (JVM) API. Therefore the JVM code exists in the same process space as the ActiveX application (Visual Basic, VBScript, or Active Server Pages (ASP) files) and remains attached to the process until that process terminates.

In the *Applet client* model, a Java applet embeds in a HyperText Markup Language (HTML) document residing on a remote client machine from the WebSphere Application Server. With this type of client, the user accesses an enterprise bean in the WebSphere Application Server through the Java applet in the HTML document.

The *J2EE application client* is a Java application program that accesses enterprise beans, Java DataBase Connectivity (JDBC) APIs, and Java Message Service message queues. The J2EE application client program runs on client machines. This program follows the same Java programming model as other Java programs; however, the J2EE application client depends on the Application Client run time to configure its execution environment, and uses the Java Naming and Directory Interface (JNDI) name space to access resources.

The *Pluggable and Thin application clients* provide a lightweight Java client programming model. These clients are useful in situations where a Java client application exists but the application needs enhancements to use enterprise beans, or where the client application requires a thinner, more lightweight environment than the one offered by the J2EE application client. The difference between the Thin application client and the Pluggable application client is that the Thin application client includes a Java virtual machine (JVM) API, and the Pluggable application client requires the user to provide this code. The Pluggable application client uses the Sun Java Development Kit, and the Thin application client uses the IBM Developer Kit for the Java platform.

The J2EE application client programming model provides the benefits of the J2EE platform for the Java client application. Use the J2EE application client to develop, assemble, deploy and launch a client application. The tooling provided with the WebSphere platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

When you develop a client application using and adhering to the J2EE platform, you can put the client application code from one J2EE platform implementation to another. The client application package can require redeployment using each J2EE platform deployment tool, but the code that comprises the client application remains the same.

The Application Client run time supplies a container that provides access to system services for the client application code. The client application code must contain a main method. The Application Client run time invokes this main method after the environment initializes and runs until the Java virtual machine code terminates.

The J2EE platform supports the Application Client use of *nicknames* or *short names*, defined within the client application deployment descriptor. These deployment descriptors identify enterprise beans or local resources (JDBC, Java Message Service (JMS), JavaMail and URL APIs) for simplified resolution through JNDI. This simplified resolution to the enterprise bean reference and local resource reference also eliminates changes to the client application code, when the underlying object or resource either changes or moves to a different server. When these changes occur, the Application Client can require redeployment.

The Application Client also provides initialization of the run-time environment for the client application. The deployment descriptor defines this unique initialization for each client application. The Application Client run time also provides support for security authentication to enterprise beans and local resources.

The Application Client uses the Java Remote Method Invocation-Internet InterORB Protocol (RMI-IIOP). Using this protocol enables the client application to access enterprise bean references and to use Common Object Request Broker Architecture (CORBA) services provided by the J2EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both enterprise bean references and CORBA object references.

When you combine the J2EE and CORBA environments or programming models in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

View the Samples gallery for more information about the Application Client.

Application client functions

This topic provides information about available functions in the different types of clients.

Use the following table to identify the available functions in the different types of clients.

Available functions	ActiveX client	Applet client	J2EE client	Pluggable client	Thin client
Provides all the benefits of a J2EE platform	Yes	No	Yes	No	No
Portable across all J2EE platforms	No	No	Yes	No	No
Provides the necessary run-time support for communication between a client and a server	Yes	Yes	Yes	Yes	Yes
Supports the use of nicknames in the deployment descriptor files. **Note:** Although you can edit deployment descriptor files, do not use the administrative console to modify them.	Yes	No	Yes	No	No
Supports use of the RMI-IIOP protocol	Yes	Yes	Yes	Yes	Yes
Browser-based application	No	Yes	No	No	No
Enables development of client applications that can access enterprise bean references and CORBA object references	Yes	Yes	Yes	Yes	Yes
Enables the initialization of the client application run-time environment	Yes	No	Yes	No	No
Supports security authentication to enterprise beans	Yes	Limited	Yes	Yes	Yes
Supports security authentication to local resources	Yes	No	Yes	No	No
Requires distribution of application to client machines	Yes	No	Yes	Yes	Yes
Enables access to enterprise beans and other Java classes through Visual Basic, VBScript, and Active Server Pages (ASP) code	Yes	No	No	No	No

Provides a lightweight client suitable for download	No	Yes	No	Yes	Yes
Enables access JNDI APIs for enterprise bean resolution	Yes	Yes	Yes	Yes	Yes
Runs on client machines that use the Sun Java Runtime Environment	No	No	No	Yes	No
Supports CORBA services (using CORBA services can render the application client code nonportable)	No	No	Yes	No	No
Supports JMS connections to the default messaging provider	No	No	Yes	Yes	Yes

ActiveX application clients

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

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

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

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

The ActiveX to EJB bridge uses the Java Native Interface (JNI) architecture to programmatically access the JVM code. Therefore the JVM code exists in the same process space as the ActiveX application (Visual Basic, VBScript, or ASP) and remains attached to the process until that process terminates. To create JVM code, an ActiveX client program calls the XJBInit() method of the XJB.JClassFactory object. For more information about creating JVM code for an ActiveX program, see ActiveX to EJB bridge, initializing JVM code.

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

The client program performs primitive data type conversion through the COM IDispatch interface (use of the IUnknown interface is not directly supported). Primitive data types are automatically converted between native automation types and Java types. All other types are handled automatically by the proxy objects For more information about data type conversion, see ActiveX to EJB bridge, converting data types.

Any exceptions thrown in Java code are encapsulated and thrown again as a COM error, from which the ActiveX program can determine the actual Java exceptions. For more information about handling exceptions, see ActiveX to EJB bridge, handling errors.

The ActiveX to EJB bridge supports both free-threaded and apartment-threaded access and implements the free threaded marshaler (FTM) to work in a hybrid environment such as Active Server Pages. For more information about the support for threading, see ActiveX to EJB bridge, using threading.

Applet clients

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

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

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

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

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

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

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

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

J2EE application clients

The J2EE application client programming model provides the benefits of the Java 2 Platform for WebSphere Application Server Enterprise product.

The J2EE platform offers the ability to seamlessly develop, assemble, deploy and launch a client application. The tooling provided with the WebSphere platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

When you develop a client application using and adhering to the J2EE platform, you can put the client application code from one J2EE platform implementation to another. The client application package can require redeployment using each J2EE platform deployment tool, but the code that comprises the client application does not change.

The J2EE application client run time supplies a container that provides access to system services for the application client code. The J2EE application client code must contain a main method. The J2EE application client run time invokes this main method after the environment initializes and runs until the Java virtual machine application terminates.

Application clients can use *nicknames* or *short names*, defined within the client application deployment descriptor with the J2EE platform. These deployment descriptors identify enterprise beans or local resources (JDBC data sources, J2C connection factories, Java Message Service (JMS), JavaMail and URL APIs) for simplified resolution through JNDI use. This simplified resolution to the enterprise bean reference and local resource reference also eliminates changes to the application client code, when the underlying object or resource either changes or moves to a different server. When these changes occur, the application client can require redeployment. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

The J2EE application client also provides initialization of the run-time environment for the client application. The deployment descriptor defines this unique initialization for each client application. The J2EE application client run time also provides support for security authentication to the enterprise beans and local resources.

The J2EE application client uses the Java Remote Method Invocation technology run over Internet Inter-Orb Protocol (RMI-IIOP). Using this protocol enables the client application to access enterprise bean references and to use Common Object Request Broker Architecture (CORBA) services provided by the J2EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both enterprise bean references and CORBA object references.

When you combine the J2EE and the CORBA WebSphere Application Server Enterprise environments or programming models in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

Pluggable application clients

The Pluggable application client provides a lightweight, downloadable Java application run time capable of interacting with enterprise beans.

The Pluggable application client requires that you have previously installed the Sun Java Runtime Environment (JRE) files. In all other aspects, the Pluggable application client, and the Thin application client are similar.

Note: The Pluggable application client is only available on the Windows platform.

This client is designed to support those users who want a lightweight Java client application programming environment, without the overhead of the J2EE platform on the client machine. The programming model for this client is heavily influenced by the CORBA programming model, but supports access to enterprise beans.

When accessing enterprise beans from this client, the client application can consider the enterprise beans object references as CORBA object references.

Tooling does not exist on the client; however, tooling does exists on the server. You are responsible for developing the client application, generating the necessary client bindings for the enterprise bean and CORBA objects, and after bundling these pieces together, installing them on the client machine.

The Pluggable application client provides the necessary run time to support the communication needs between the client and the server.

The Pluggable application client uses the RMI-IIOP protocol. Using this protocol enables the client application to access enterprise bean references and CORBA object references and use any supported CORBA services. Using the RMI-IIOP protocol along with the accessibility of CORBA services can assist a user in developing a client application that needs to access both enterprise bean references and CORBA object references.

When you combine the J2EE and CORBA environments in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

The Pluggable application client run time provides the necessary support for the client application for object resolution, security, Reliability Availability and Serviceability (RAS), and other services. However, this client does not support a container that provides easy access to these services. For example, no support exists for using *nicknames* for enterprise beans or local resource resolution. When resolving to an enterprise bean (using either the Java Naming and Directory Interface (JNDI) API or CosNaming) sources, the client application must know the location of the name server and the fully qualified name used when the reference was bound into the name space.

When resolving to a local resource, the client application cannot resolve to the resource through a JNDI lookup. Instead the client application must explicitly create the connection to the resource using the appropriate API (JDBC, Java Message Service (JMS), and so on). This client does not perform initialization of any of the services that the client application might require. For example, the client application is responsible for the initialization of the naming service, either through CosNaming or JNDI APIs.

The Pluggable application client offers access to most of the available client services in the J2EE application client. However, you cannot access the services in the Pluggable application client as easily as you can in the J2EE application client. The J2EE client has the advantage of performing a simple Java Naming and Directory Interface (JNDI) name space lookup to access the desired service or resource. The Pluggable application client must code explicitly for each resource in the client application. For example, looking up an enterprise bean Home object requires the following code in a J2EE application client:

```
        java.lang.Object ejbHome =  initialContext.lookup("java:/comp/env/ejb/MyEJBHome"
);
    MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,
MyEJBHome.class);
```

However, you need more explicit code in a Pluggable application client for Java:

```
        java.lang.Object ejbHome =  initialContext.lookup("the/fully/qualified
/path/to/actual/home/in/namespace/MyEJBHome");
    MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,
MyEJBHome.class);
```

In this example, the J2EE application client accesses a logical name from the `java:/comp` name space. The J2EE client run time resolves that name to the physical location and returns the reference to the client application. The pluggable client must know the fully qualified physical location of the enterprise bean Home object in the name space. If this location changes, the pluggable client application must also change the value placed on the lookup() statement.

In the J2EE client, the client application is protected from these changes because it uses the logical name. A change can require a redeployment of the EAR file, but the actual client application code remains the same.

The Pluggable application client is a traditional Java application that contains a *main* function. The WebSphere Pluggable application client provides run-time support for accessing remote enterprise beans, and provides the implementation for various services (security, Workload Management (WLM), and others). This client can also access CORBA objects and CORBA-based services. When using both

environments in one client application, you need to understand the differences between the enterprise bean and the CORBA programming models to manage both environments.

For instance, the CORBA programming model requires the CORBA CosNaming name service for object resolution in a name space. The enterprise beans programming model requires the JNDI name service. The client application must initialize and properly manage these two naming services.

Another difference applies to the enterprise bean model. Use the Java Naming and Directory Interface (JNDI) implementation in the enterprise bean model to initialize the Object Request Broker (ORB). The client application is unaware that an ORB is present. The CORBA model, however, requires the client application to explicitly initialize the ORB through the `ORB.init()` static method.

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

Thin application clients

The thin application client provides a lightweight, downloadable Java application run time capable of interacting with enterprise beans.

WebSphere Application Server Version 6.1 supports the pluggable client.

The thin client is designed to support those users who want a lightweight Java client application programming environment, without the overhead of the J2EE platform on the client machine. The programming model for this client is heavily influenced by the CORBA programming model, but supports access to enterprise beans.

When accessing enterprise beans from this client, the client application can consider the enterprise beans object references as CORBA object references.

Tooling does not exist on the client, it exists on the server. You are responsible for developing the client application, generating the necessary client bindings for the enterprise bean and CORBA objects, and bundling these pieces together to install on the client machine.

The thin application client provides the necessary runtime to support the communication needs between the client and the server.

The thin application client uses the RMI-IIOP protocol. Using this protocol enables the client application to access not only enterprise bean references and CORBA object references, but also allows the client application to use any supported CORBA services. Using the RMI-IIOP protocol along with the accessibility of CORBA services can assist a user in developing a client application that needs to access both enterprise bean references and CORBA object references.

When you combine the J2EE and CORBA environments in one client application, you must understand the differences between the two programming models, to use and manage each appropriately.

The thin application client run time provides the necessary support for the client application for object resolution, security, Reliability Availability and Servicability (RAS), and other services. However, this client does not support a container that provides easy access to these services. For example, no support exists for using *nicknames* for enterprise beans or local resource resolution. When resolving to an enterprise bean (using either Java Naming and Directory Interface (JNDI) or CosNaming) sources, the client application must know the location of the name server and the fully qualified name used when the reference was bound into the name space. When resolving to a local resource, the client application cannot resolve to the resource through a JNDI lookup. Instead the client application must explicitly create the connection to the resource using the appropriate API (JDBC, Java Message Service (JMS), and so

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

The thin application client offers access to most of the available client services in the J2EE application client. However, you cannot access the services in the thin client as easily as you can in the J2EE application client. The J2EE client has the advantage of performing a simple Java Naming and Directory Interface (JNDI) name space lookup to access the desired service or resource. The thin client must code explicitly for each resource in the client application. For example, looking up an enterprise bean Home requires the following code in a J2EE application client:

```
        java.lang.Object ejbHome =  initialContext.lookup("java:/comp/env/ejb/MyEJBHome");
    MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,  MyEJBHome.class);
```

However, you need more explicit code in a Java thin application client:

```
        java.lang.Object ejbHome =
initialContext.lookup("the/fully/qualified/path/to/actual/home/in/namespace/MyEJBHome");
    MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,  MyEJBHome.class);
```

In this example, the J2EE application client accesses a logical name from the `java:/comp` name space. The J2EE client run time resolves that name to the physical location and returns the reference to the client application. The thin client must know the fully qualified physical location of the enterprise bean Home in the name space. If this location changes, the thin client application must also change the value placed on the lookup() statement.

In the J2EE client, the client application is protected from these changes because it uses the logical name. A change might require a redeployment of the EAR file, but the actual client application code remains the same.

The thin application client is a traditional Java application that contains a *main* function. The WebSphere thin application client provides run-time support for accessing remote enterprise beans, and provides the implementation for various services (security, Workload Management (WLM), and others). This client can also access CORBA objects and CORBA based services. When using both environments in one client application, you need to understand the differences between the enterprise bean and CORBA programming models to manage both environments.

For instance, the CORBA programming model requires the CORBA CosNaming name service for object resolution in a name space. The enterprise beans programming model requires the JNDI name service. The client application must initialize and properly manage these two naming services.

Another difference applies to the enterprise bean model. Use the Java Naming and Directory Interface (JNDI) implementation in the enterprise bean model to initialize the Object Request Broker (ORB). The client application is unaware that an ORB is present. The CORBA model, however, requires the client application to explicitly initialize the ORB through the `ORB.init()` static method.

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

Application client troubleshooting tips

This topic provides debugging tips for resolving common Java 2 Platform Enterprise Edition (J2EE) application client problems. To use this troubleshooting guide, review the trace entries for one of the J2EE application client exceptions, and then locate the exception in the guide.

Some of the errors in the guide are samples, and the actual error you receive can be different than what is shown here. You might find it useful to rerun the `launchClient` command specifying the `-CCverbose=true` option. This option provides additional information when the J2EE application client run time is initializing.

Error: java.lang.NoClassDefFoundError

Explanation

This exception is thrown when Java code cannot load the specified class.

Possible causes

- Invalid or non-existent class
- Class path problem
- Manifest problem

Recommended response

Check to determine if the specified class exists in a Java Archive (JAR) file within your Enterprise Archive (EAR) file. If it does, make sure the path for the class is correct. For example, if you get the exception:

```
java.lang.NoClassDefFoundError:
WebSphereSamples.HelloEJB.HelloHome
```

verify that the HelloHome class exists in one of the JAR files in your EAR file. If it exists, verify that the path for the class is WebSphereSamples.HelloEJB.

If both the class and path are correct, then it is a class path issue. Most likely, you do not have the failing class JAR file specified in the client JAR file manifest. To verify this situation, perform the following steps:

1. Open your EAR file with the Application Server Toolkit or the Rational Web Developer assembly tool, and select the Application Client.
2. Add the names of the other JAR files in the EAR file to the Classpath field.

This exception is generally caused by a missing Enterprise Java Beans (EJB) module name from the Classpath field.

If you have multiple JAR files to enter in the Classpath field, be sure to separate the JAR names with spaces.

If you still have the problem, you have a situation where a class is loaded from the file system instead of the EAR file. This error is difficult to debug because the offending class is not the one specified in the exception. Instead, another class is loaded from the file system before the one specified in the exception. To correct this error, review the class paths specified with the -CCclasspath option and the class paths configured with the Application Client Resource Configuration Tool. Look for classes that also exist in the EAR file. You must resolve the situation where one of the classes is found on the file system instead of in the `.ear` file. Remove entries from the classpaths, or include the `.jar` files and classes in the `.ear` file instead of referencing them from the file system.

If you use the -CCclasspath parameter or resource classpaths in the Application Client Resource Configuration Tool, and you have configured multiple JAR files or classes, verify they are separated with the correct character for your operating system. Unlike the Classpath field, these class path fields use platform-specific separator characters, usually a colon (on operating systems such as AIX or Linux) or a semi-colon (on Windows systems).
Note: The system class path is not used by the Application Client run time if you use the launchClient batch or shell files. In this case, the system class path would not cause this problem. However, if you load the launchClient class directly, you do have to search through the system class path as well.

Error: com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred while attempting to get an instance of the object for the specified reference object. [Root exception is javax.naming.NameNotFoundException: xxxxxxxxxx]

Explanation

This exception occurs when you perform a lookup on an object that is not installed on the host server. Your program can look up the name in the local client Java Naming and Directory Interface (JNDI) name space, but received a NameNotFoundException exception because it is not located on the host server. One typical example is looking up an EJB component that is not installed on the host server that you access. This exception might also occur if the JNDI name you configured in your Application Client module does not match the actual JNDI name of the resource on the host server.

Possible causes

- Incorrect host server invoked
- Resource is not defined
- Resource is not installed
- Application server is not started
- Invalid JNDI configuration

Recommended response

If you are accessing the wrong host server, run the `launchClient` command again with the -CCBootstrapHost parameter specifying the correct host server name. If you are accessing the correct host server, use the product `dumpnamespace` command line tool to see a listing of the host server JNDI name space. If you do not see the failing object name, the resource is either not installed on the host server or the appropriate application server is not started. If you determine the resource is already installed and started, your JNDI name in your client application does not match the global JNDI name on the host server. Use the Application Server Toolkit to compare the JNDI bindings value of the failing object name in the client application to the JNDI bindings value of the object in the host server application. The values must match.

Error: javax.naming.ServiceUnavailableException: A communication failure occurred while attempting to obtain an initial context using the provider url: ″iiop://[invalidhostname]″. Make sure that the host and port information is correct and that the server identified by the provider URL is a running name server. If no port number is specified, the default port number 2809 is used. Other possible causes include the network environment or workstation network configuration. Root exception is org.omg.CORBA.INTERNAL: JORB0050E: In Profile.getIPAddress(), InetAddress.getByName[invalidhostname] threw an UnknownHostException. minor code: 4942F5B6 completed: Maybe

Explanation

This exception occurs when you specify an invalid host server name.

Possible causes

- Incorrect host server invoked
- Invalid host server name

Recommended response

Run the `launchClient` command again and specify the correct name of your host server with the -CCBootstrapHost parameter.

Error: javax.naming.CommunicationException: Could not obtain an initial context due to a communication failure. Since no provider URL was specified, either the bootrap host and port of an existing ORB was used, or a new ORB instance was created and initialized with the default bootstrap host of ″localhost″ and the default bootstrap port of 2809. Make sure the ORB bootstrap host and port resolve to a running name server. Root exception is org.omg.CORBA.COMM_FAILURE: WRITE_ERROR_SEND_1 minor code: 49421050 completed: No

Explanation

This exception occurs when you run the `launchClient` command to a host server that does not have the Application Server started. You also receive this exception when you specify an invalid host server name. This situation might occur if you do not specify a host server name when you run the launchClient tool. The default behavior is for the launchClient tool to run to the local host, because WebSphere Application Server does not know the name of your host server. This default behavior only works when you are running the client on the same machine with WebSphere Application Server is installed.

Possible causes

- Incorrect host server invoked
- Invalid host server name
- Invalid reference to `localhost`
- Application server is not started
- Invalid bootstrap port

Recommended response

If you are not running to the correct host server, run the `launchClient` command again and specify the name of your host server with the `-CCBootstrapHost` parameter. Otherwise, start the Application Server on the host server and run the `launchClient` command again.

Error: javax.naming.NameNotFoundException: Name comp/env/ejb not found in context ″java:″

Explanation

This exception is thrown when the Java code cannot locate the specified name in the local JNDI name space.

Possible causes

- No binding information for the specified name
- Binding information for the specified name is incorrect
- Wrong class loader was used to load one of the program classes
- A resource reference does not include any client configuration information
- A client container on the deployment manager is trying to use enterprise extensions (not supported)

Recommended response

Open the EAR file with the Application Server Toolkit, and check the bindings for the failing name. Ensure this information is correct. If you are using Resource References, open the EAR file with the Application Client Resource Configuration Tool, and verify that the Resource Reference has client configuration information and the name of the Resource Reference exactly matches the JNDI name of the client configuration. If the values are correct, you might have a class loader error. For detailed information about the configuration tool and opening EAR files, read the Starting the Application Client Resource Configuration Tool in the *Developing and deploying applications* PDF book.

Error: java.lang.ClassCastException: Unable to load class: org.omg.stub.WebSphereSamples.HelloEJB._HelloHome_Stub at com.ibm.rmi.javax.rmi.PortableRemoteObject.narrow(portableRemoteObject.java:269)

Explanation

This exception occurs when the application program attempts to narrow to the EJB home class and the class loaders cannot find the EJB client side bindings.

Possible causes

- The files, *_Stub.class and _Tie.class, are not in the EJB `.jar` file
- Class loader could not find the classes

Recommended response

Look at the EJB `.jar` file located in the `.ear` file and verify the class contains the Enterprise Java Beans (EJB) client side bindings. These are class files with file names that end in `_Stub` and `_Tie`. If the binding classes are in the EJB `.jar` file, then you might have a class loader error.

Error: WSCL0210E: The Enterprise archive file [EAR file name] could not be found. com.ibm.websphere.client.applicationclient.ClientContainerException: com.ibm.etools.archive.exception.OpenFailureException

Explanation

This error occurs when the application client run time cannot read the Enterprise Archive (EAR) file.

Possible causes

The most likely cause of this error is that the system cannot find the EAR file cannot be found in the path specified on the `launchClient` command.

Recommended response

Verify that the path and file name specified on the `launchclient` command are correct. If you are running on the Windows operating system and the path and file name are correct, use a short version of the path and file name (8 character file name and 3 character extension).

The launchClient command appears to hang and does not return to the command line when the client application has finished.

Explanation

When running your application client using the `launchClient` command the WebSphere Application Server run time might need to display the security login dialog. To display this dialog, WebSphere Application Server run time creates an Abstract Window Toolkit (AWT) thread. When your application returns from its main method to the application client run time, the application client run time attempts to return to the operating system and end the Java virtual machine (JVM) code. However, since there is an AWT thread, the JVM code will not end until `System.exit` is called.

Possible causes

The JVM code does not end because there is an AWT thread. Java code requires that `System.exit()` be called to end AWT threads.

Recommended response

- Modify your application to call `System.exit(0)` as the last statement.
- Use the `-CCexitVM=true` parameter when you call the `launchClient` command.

The applet client application client fails to launch an HTML browser in Internet Explorer

Explanation

Applet client applications run only on Windows systems. When the applet client application runs, the application output data is displayed in a browser window. If you are using Internet Explorer with the Windows XP operating system for Service Pack 2 , then you might get errors when trying to display output data.

Possible causes

The Windows XP operating system for Service Pack 2 has a security feature that blocks pop-up browser windows from appearing.

Recommended response

- Locate the information bar found under the URL Address bar in the Internet Explorer pop-up browser that has been blocked.
- Click the Information Bar to display options that disable the operating system security feature.
- Select **Allow blocked content**. You are prompted with a security window asking you to confirm your selection to allow blocked content.
- Click **Yes**.
- The applet client application runs successfully, and the browser information is displayed appropriately.

Installing the Developer Kit feature downgrades the JRE files from Version 6.0.1 or Version 6.0.2 to Version 6.0

Explanation

If you select the Developer Kit feature on the Application Client Version 6.0.0 installer, all the files are installed under the <*client_install_root*>/java directory, instead of leaving the Java Runtime Environment (JRE) files intact. The JRE files are unexpectedly downgraded to the Version 6.0.0 level.

Possible causes

Selecting the **Developer Kit** feature on the Application Client 6.0.0 installer will actually install all files under the <*client_install_root*>/java directory rather than leave the JRE files intact. Therefore, the JRE files are unexpected downgraded to the 6.0.0 level in the above installation scenario.

Recommended response

Complete the following installation steps to prevent unexpected downgrading of the JRE files.

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

- http://www.ibm.com/software/webservers/appserv/was/support/

Running application clients

The J2EE specification requires support for a client container that runs stand-alone Java applications (known as J2EE application clients) and provides J2EE services to the applications. J2EE services include naming, security, and resource connections.

You are ready to run your application client using this tool after you have:
1. Written the application client program.
2. Assembled and installed an application module (`.ear` file) in the application server run time.

3. Deployed the application using the Application Client Resource Configuration Tool (ACRCT) on Windows .

This task only applies to J2EE application clients. To launch J2EE application clients using the launchClient script, perform the following steps:

1.

   On the CL command line, enter the following command to start the Qshell environment:

   ```
   STRQSH
   ```

   a. Enter the following command to launch J2EE application clients:

      ```
      app_server_root/bin/launchClient
      ```

2. Pass parameters to the `launchClient` command or to your application client program as well. The `launchClient` command allows you to do both. The `launchClient` command requires that the first parameter is either:
   - An EAR file specifying the application client to launch.
   - A request for `launchClient` usage information.

   The following example illustrates the command line invocation syntax for the launchClient tool:

   ```
   launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]
   ```

   where
   - *userapp.ear* is the path and the name of the EAR file that contains the application client.
   - *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.
   - *app args* are arguments that pass to the application client.
   - *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation. The default is **default_profile**.
   - *-JVMOptions* is a valid Java standard or non-standard option string. Insert quotation marks around the string.
   - *-help, -?* prints the usage information.

   All other parameters intended for the `launchClient` command must begin with the -CC prefix.

   Parameters that are not EAR files, or usage requests, or that do not begin with the -CC prefix, are ignored by the application client run time, and are passed directly to the application client program.

   The `launchClient` command retrieves parameters from three places:
   - The command line
   - A properties file
   - System properties

   The parameters are resolved in the order listed above, with command line values having the highest priority and system properties the lowest. Using this prioritization you can set and override default values.

3. Specify the server name.

   By default, the **launchClient** command uses the localhost for the `BootstrapHost` property value.

   This setting is effective for testing your application client when it is installed on the same computer as the server. However, in other cases override this value with the name of your server. You can override the `BootstrapHost` value by invoking `launchClient` command with the following parameters:

   ```
   launchClient myapp.ear -CCBootstrapHost=abc.midwest.mycompany.com
   ```

   You can also override the default by specifying the value in a properties file and passing the file name to the launchClient shell.

   Security is controlled by the server. You do not need to configure security on the client because the client assumes that security is enabled. If server security is not enabled, then the server ignores the security request, and the application client functions as expected.

You can store launchClient values in a properties file, which is a good method for distributing default values. You can then override one or more values on the command line. The format of the file is one `launchClient -CC` parameter per line without the `-CC` prefix. For example:

```
 verbose=true classpath=c:\mydir\util.jar;c:\mydir\harness.jar;c:\production\G19
\global.jar BootstrapHost=abc.westcoast.mycompany.com tracefile=c:\WebSphere\mylog.txt
```

launchClient tool

This topic describes the Java 2 Platform Enterprise Edition (J2EE) command line syntax for the launchClient tool for WebSphere Application Server.

The following example illustrates the command line invocation syntax for the launchClient tool:

`launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]`

where

- *userapp.ear* is the path and the name of the EAR file that contains the application client.
- *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.
- *app args* are arguments that pass to the application client.
- *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation.

  The default is **default_profile**.
- *-JVMOptions* is a valid Java standard or nonstandard option string. Insert quotation marks around the string.
- *-help, -?* prints the usage information.

The first parameter must be `-help`, `-?` or contain no parameter at all. The `-profileName pName` and `-JVMOptions` options are optional parameters. If used, they must appear before the *<userapp>* parameter. All other parameters are optional and can appear in any order after the *<userapp>* parameter. The J2EE Application client run time ignores any optional parameters that do not begin with a `-CC` prefix and passes those parameters to the application client.

Client container parameters

Supported arguments include:

-CCadminConnectorHost
Specifies the host name of the server from which configuration information is retrieved.

The default is the value of the `-CCBootstrapHost` parameter or the value, `localhost`, if the `-CCBootstrapHost` parameter is not specified.

-CCadminConnectorPort
Indicates the port number for the administrative client function to use. The default value is `8880` for SOAP connections and `2809` for Remote Method Invocation (RMI) connections.

-CCadminConnectorType
Specifies how the administrative client connects to the server. Specify `RMI` to use the RMI connection type, or specify `SOAP` to use the SOAP connection type. The default value is `SOAP`.

-CCadminConnectorUser
Administrative clients use this user name when a server requires authentication. If the connection type is SOAP, and security is enabled on the server, this parameter is required. The SOAP connector does not prompt for authentication.

-CCadminConnectorPassword
The password for the user name that the `-CCadminConnectorUser` parameter specifies.

-CCaltDD
The name of an alternate deployment descriptor file. This parameter is used with the `-CCjar` parameter to specify the deployment descriptor to use. Use this argument when a client JAR file is configured with more than one deployment descriptor. Set the value to `null` to use the client JAR file standard deployment descriptor.

-CCBootstrapHost
The name of the host server you want to connect to initially. The format is: *your_server_of_choice.com*

-CCBootstrapPort
The server port number. If you do not specify this argument, the WebSphere Application Server default value is `used`.

-CCclassLoaderMode
Specifies the class loader mode. If PARENT_LAST is specified, the class loader loads classes from the local class path before delegating the class loading to its parent. The classes loaded for the following are affected:

- Classes defined for the J2EE application client
- Resources defined in the J2EE application
- Classes specified on the manifest of the J2EE client JAR file
- Classes specified using the -CCclasspath option

If PARENT_LAST is not specified, then the default mode, PARENT_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.

-CCclasspath
A class path value. When you launch an application, the system class path is used. If you want to access classes that are not in the EAR file or part of the system class paths, specify the appropriate class path here. Multiple paths can be concatenated.

-CCD
Use this option to have the WebSphere Application Server set the specified system property during initialization. Do not use the equals (=) character after the `-CCD`. For example: `-CCDcom.ibm.test.property=testvalue`. You can specify multiple `-CCD` parameters. The general format of this parameter is `-CCD<property key>=<property value>`. For example, `-CCDI18NService.enable=true`.

-CCdumpJavaNameSpace
Prints out the Java portion of the Java Naming and Directory Interface (JNDI) name space for WebSphere Application Server. The `true` value uses the short format that prints out the binding name and the type of the object bound at that location. The `long` value uses the long format that prints out the binding name, bound object type, local object, type and string representation of the local object, for example, IORs and string values. The default value is `false`.

-CCexitVM
Use this option to have the WebSphere Application Server call the `System.exit()` method after the client application completes. The default is `false`.

-CCinitonly
Use this option to initialize application client run time for ActiveX application clients without launching the client application. The default is `false`.

-CCjar
The name of the client Java Archive (JAR) file that resides within the EAR file for the application you wish to launch. Use this argument when you have multiple client JAR files in the EAR file.

-CCpropfile
Indicates the name of a properties file that contains launchClient properties. Specify the properties

without the -CC prefix in the file, with the exception of the securityManager, securityMgrClass and securityMgrPolicy properties. See the following example: verbose=true.

-CCproviderURL
Provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a Common Object Request Broker Architecture (CORBA) object URL or an Internet Inter-ORB Protocol (IIOP) URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. You can specify bootstrap server addresses, for all servers in the cluster, in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the -CCBootstrapHost and -CCBootstrapPort parameters. A CORBA object URL specifying multiple systems is illustrated in the following example:

```
-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809
```

This value is mapped to the java.naming.provider.url system property.

-CCsecurityManager
Enables and runs the WebSphere Application Server with a security manager. The default is disable.

-CCsecurityMgrClass
Indicates the fully qualified name of a class that implements a security manager. Only use this argument if the -CCsecurityManager parameter is set to enable. The default is java.lang.SecurityManager.

-CCsecurityMgrPolicy
Indicates the name of a security manager policy file. Only use this argument if the -CCsecurityManager parameter is set to enable. When you enable this parameter, the java.security.policy system property is set. The default is *<app_server_root>*/ properties/client.policy.

-CCsoapConnectorPort
The Simple Object Access Protocol (SOAP) connector port. If you do not specify this argument, the WebSphere Application Server default value is used.

-CCtrace
Use this option to obtain debug trace information. You might need this information when reporting a problem to IBM customer support. The default is false. For more information, read the topic ″Enabling trace.″

-CCtracefile
Indicates the name of the file to which trace information is written. The default is to write output to the console.

-CCtraceMode
Specifies the trace format to use for tracing. If the valid value, basic, is not specified the default is advanced. Basic tracing format is a more compact form of tracing.

For more information on basic and advanced trace formatting, see Interpreting trace output.

-CCverbose
This option displays additional information messages. The default is false.

The following examples demonstrate correct syntax.

Windows

```
launchClient c:\earfiles\myapp.ear -CCBootstrapHost=myWASServer -CCverbose=true app_parm1 app_parm2
```

Specifying the directory for an expanded EAR file

You can archive the `Manifest.mf` client Java Archive (JAR) files instead of automatically cleaning them up after the application exits.

Each time the launchClient tool is called, it extracts the Enterprise Archive (EAR) file to a random directory name in the temporary directory on your hard drive. Then the tool sets up the thread ClassLoader to use the extracted EAR file directory and JAR files included in the `Manifest.mf` client Java Archive (JAR) file. In a normal J2EE Java client, these files are automatically cleaned up after the application exits. This cleanup occurs when the client container shutdown hook is called. To avoid extracting the EAR file (and removing the temporary directory) each time the launchClient tool is called, complete the following steps:

1. Specify a directory to extract the EAR file by setting the `com.ibm.websphere.client.applicationclient.archivedir` Java system property. If the directory does not exist or is empty, the EAR file is extracted normally. If the EAR file was previously extracted, the launchClient tool reuses the directory.
2. Delete the directory before running the launchClient tool again, if you need to update your EAR file. When you call the `launchClient` command, it extracts the new EAR file to the directory. If you do not delete the directory or change the system property value to point to a different directory, the launchClient tool reuses the currently extracted EAR file and does not use your changed EAR file. When specifying the `com.ibm.websphere.client.applicationclient.archivedir` property, make sure that the directory you specify is unique for each EAR file you use. For example, do not point the `MyEar1.ear` and the `MyEar2.ear` files to the same directory.

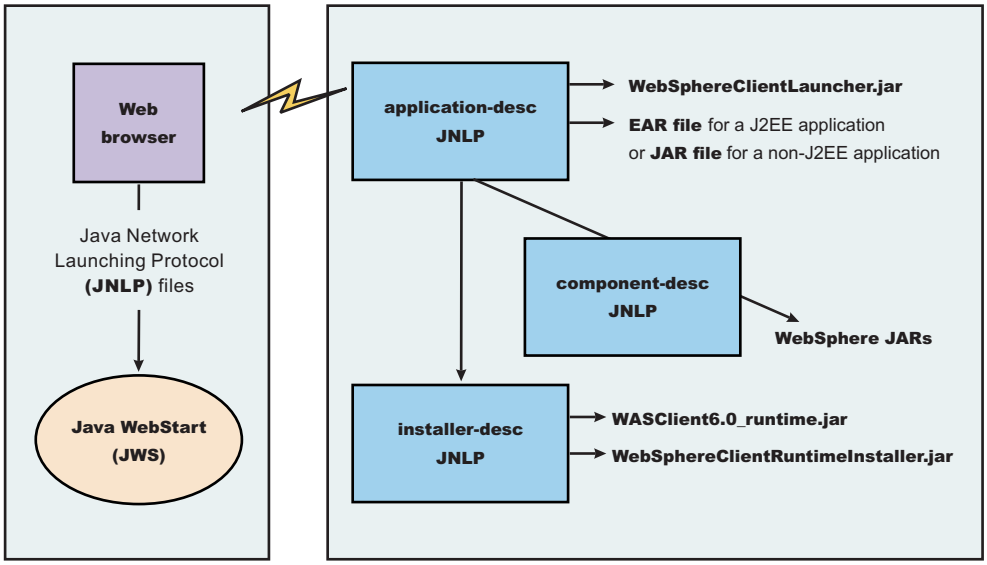
Java Web Start architecture for deploying application clients

Java Web Start is an application-deployment technology that includes the portability of applets, the maintainability of servlets and JavaServer Pages (JSP) file technology, and the simplicity of mark-up languages such as XML and HTML. It is a Java application that allows full-featured Java 2 client applications to be launched, deployed and updated from a standard Web server. The Java Web Start client is used with platforms that support a Web browser.

Upon launching Java Web Start for the first time, you might download new client applications from the Web. Each time you launch JWS thereafter, you can initiate applications either through a link on a Web page or (in Windows) from desktop icons or the Start menu. You can deploy applications quickly using Java Web Start, cache applications on the client machine, and launch applications remotely offline. Additionally, because Java Web Start is built from the J2EE infrastructure, the technology inherits the complete security architecture of the J2EE platform.

The technology underlying Java Web Start is the Java Network Launching Protocol & API (JNLP). Java Web Start is a JNLP client and it reads and parses a JNLP descriptor file (JNLP file). Base on the JNLP descriptor, it downloads appropriate pieces of a client application and any of its dependencies. If any of the pieces of the application are already cached on the client machine, then those components are not downloaded again, unless they have been updated on the server machine. After you download and cache the client application, JWS launches it natively on the client machine.

The following diagram shows an overview of launching a client application, include the Application Client for WebSphere Application Server, Version 6 as a dependent resource, using Java Web Start.



The Web browser running on a client machine connects to a Web application located on a server machine. The client application JNLP descriptor file is downloaded and processed by Java Web Start on the client machine.

In this diagram, there are three JNLP descriptor files:

- Client application JNLP descriptor (application-desc in the diagram)
- Application Clients run-time installer JNLP descriptor (installer-desc in the diagram)
- Application Clients run-time library component JNLP descriptor (component-desc in the diagram)

Each of these JNLP descriptor files, the client application (JAR or EAR) and the dependent resource JAR files are packaged as Web applications in an EAR file. This EAR file is deployed to an Application server. The client machine with JWS installed uses a Web browser to connect to the url of the client application JNLP descriptor file to download and run the client application.

Using Java Web Start from J2SE Java Runtime Environment 5.0 or later is highly recommended. All the platforms supported by the application client for WebSphere Application Server are supported with the exception Linux on Power and OS400 platforms.

You can use any of the following:

- Java Web Start on the Java 2 Standard Edition Developer Kits that IBM provides, packaged in Application Client for WebSphere Application Server, Version 6.1
- Java Web Start on Sun Microsystems J2SE Software Development Kit or J2SE Java Runtime Environment 5.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems
- Java Web Start on HP-UX JDK or JRE for Java 2 Platform Standard Edition, version 5, which you can download from the HP Web site

Using Java Web Start

This topic provides the steps and prerequisites necessary to use Java Web start.

Before you begin this task, see the following topics to understand Java Web Start technology and its components:

- “Java Web Start architecture for deploying application clients” on page 247

- “Client application Java Network Launcher Protocol deployment descriptor file” on page 250
- “ClientLauncher class” on page 254

Note: You can use any of the following:

- Java Web Start on Java 2 Standard Edition Developer Kits that IBM provides, packaged in the Application Client for WebSphere Application Server, Version 6.1
- Java Web Start on Sun Microsystems J2SE Software Development Kit or J2SE Java Runtime Environment 5.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems
- Java Web Start on HP-UX JDK or JRE for Java 2 Platform Standard Edition, version 5.0, which you can download from the HP Web site.

1. Prepare the Application Clients run-time dependency component for JWS.
2. Prepare the Application Clients run-time library component for JWS.
3. Installing JWS.
4. **Optional:** Run the Java Web Start sample.

   Problem: When you run Web services clients from Java Web Start using a Mozilla browser, you might get errors if the client argument contains quotations in the jnlp.jsp file. For example, the following argument results in an error:

   ```
   <argument>-url="wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
   /pli/ejb/WSMultiProtocolHome&"</argument>
   ```

   Error: The following errors display in the Java Web Start console:

   If using the EJB protocol, the following error is displayed:

   ```
   Client caught exception getting the InsuranceWebServicesPort
   using the URL
   "wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
   /pli/ejb/WSMultiProtocolHome&"
   java.net.MalformedURLException: no protocol:
   "wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
   /pli/ejb/WSMultiProtocolHome&"
    at java.net.URL.<init>(URL.java(Compiled Code))
    at java.net.URL.<init>(URL.java(Compiled Code))
    at java.net.URL.<init>(URL.java:411)
    at com.ibm.wssvt.tc.pli.webservice.InsuranceWebServicesClient
   .getInsuranceServicesClientURL(InsuranceWebServicesClient.java:231)
    at com.ibm.wssvt.tc.pli.webservice.InsuranceWebServicesClient
   .main(InsuranceWebServicesClient.java:748)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:85)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:58)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:60)
    at java.lang.reflect.Method.invoke(Method.java:391)
    at com.ibm.websphere.client.applicationclient.launchClient.
   createContainerAndLaunchApp(launchClient.java:649)
   ```

   If using the HTTP protocol, the following error is displayed:

   ```
   Client caught exception getting the InsruanceWebServicesPort
   using the URL
   "http://svtlnx1:9081/WebSvcsInsSession20EJB/services/WSMultiProtocol"
   java.net.MalformedURLException: no protocol:
   "http://svtlnx1:9081/WebSvcsInsSession20EJB/services/WSMultiProtocol"
   ```

   If using the JMS protocol, the following error is displayed:

   ```
   Client caught exception getting the InsruanceWebServicesPort
   using the URL
   "jms:/queue?destination=jms/MultiProtocol_Q&connectionFactory=jms/InsuranceServices_Q

   CF&targetService=WSMultiProtocolJMS&jndiProviderURL=IIOP://svtlnx1.austin.ibm.com:981
   1"
   ```

```
java.net.MalformedURLException: no protocol:
"jms:/queue?destination=jms/MultiProtocol_Q&connectionFactory=jms/InsuranceServices_Q

CF&targetService=WSMultiProtocolJMS&jndiProviderURL=IIOP://svtlnx1.austin.ibm.com:981
1"
       at java.net.URL.<init> (URL.java(Compiled Code))
Making calls to methods in WSMultiprotocolWebServicesBean ...
```

Solution: To resolve the problem, update the jnlp.jsp file to remove the quotations (″ ″) from the argument. When a jnlp.jsp file has statements or expressions that begin with ″${″ and the statement is not to be interpreted as an expression, then add a backward slash ″\″, as shown in the following example:

```
<argument>-CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/DummyClientKeyFile.jks</argument>
<argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/DummyClientTrustFile.jks</argument>
```

For the EJB protocol, use the following example argument to correct the errors:

```
<argument>-url=wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
/pli/ejb/WSMultiProtocolHome&</argument>
```

For the HTTP protocol, use the following argument to correct the errors:

```
<argument>-url=http://svtaix23:9081/WebSvcsInsSession20EJB/services/WSMultiProtocol</argument>
```

For the JMS protocol, use the following argument to correct the errors:

```
<argument>-url=jms:/queue?destination=jms/MultiProtocol_Q&connectionFactory=
jms/InsuranceServices_QCF&targetService=
WSMultiProtocolJMS&jndiProviderURL=IIOP://svtaix23.austin.ibm.com:9811</argument>
```

Now, rerun the client from Java Web Start.

Client application Java Network Launcher Protocol deployment descriptor file

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application.

Location

The client application has an Application Clients run-time dependency that provides the following:

- Java 2 Runtime Environment from IBM
- Application Clients run-time properties
- SSL KeyStore and TrustStore file
- Application Clients run-time library JAR files (optional for Thin Application client applications)

If the Application Clients run-time dependency is not met, it is downloaded and installed in Java Web Start (JWS), as described by the Application Clients run-time installer JNLP descriptor file.

```
<j2se version="WASclient6.1.0"
href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJre/AppClientRT.jsp"/>
```

Usage notes

The client application must also include the `WebSphereClientLauncher.jar` file, which contains the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, that completes one of the following actions:

- If it is a J2EE Application client application (that is the resources for the application contain an EAR file with a client application), then the launcher class starts a second Java Virtual Machine (JVM) using the JRE provided by the Application Clients run-time dependency and launches the J2EE Application client application that is packaged in the EAR file.

  The EAR file must be specified as a JAR resource so that it can be downloaded to JWS and specified in the system property, `com.ibm.websphere.client.launcher.ear`. See “JNLP descriptor file for a J2EE Application client application” on page 251 for an example.

- If it is a Thin Application client application, then the launcher class uses the current JVM from the Application Clients run-time dependency and invokes the Thin Application client application main method.

  The Thin Application client application JAR file must be specified as a JAR resource so that it can be downloaded to JWS and the name of the class containing main method entry point is specified in the system property, `com.ibm.websphere.launcher.main`. See "JNLP descriptor file for a Thin Application client application" on page 253 for an example.

  Unlike the J2EE Application client application, the Thin Application client application is not loading the Application Clients run-time library JAR files from the Application Clients run-time dependency. It is downloaded from the server directly as it is for the Thin Application client application JAR file. An Application Clients run-time library component JNLP descriptor is used for specifying the Application Clients run-time library JAR files resources, as shown in the following example:

  ```
  <extension name="WAS Thin EJB Client Library"
                    href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/AppClientLib.jnlp"/>
  ```

  The JNLP specification requires all the resource (JAR or EAR) files used in a JNLP file to be signed.

  You can specify the –CC arguments defined in the launchClient tool for a J2EE Application client application in application arguments section of the JNLP descriptor files. However, only –CCD is supported for a Thin Application client application to define system properties and the JNLP <property> tag can also be used to define system properties. See the following example for details:

  ```
  <property name="java.naming.provider.url" value="corbaloc:iiop:myserver.com:9089"/>
  ```

  For a J2EE Application client application, specify the following application arguments as defined in the JNLP.

  1. Specify your target server provider URL, as shown in the following example:

     ```
     <argument> >-CCDjava.naming.provider.url =corbaloc:iiop:myserver.mydomain.com:9080 </argument>
     ```

  2. Specify the SSL Key File and SSL Trust File location. These files are expected to be available in the client machine. To use the ones in the Application Clients run-time dependency installed in JWS cache, specify these application arguments:

     ```
     <argument> -CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12 </argument>
     <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12 </argument>
     ```

  3. Specify the initial naming context factor, as shown in the following example:

     ```
     <argument>-CCDjava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory </argument>
     ```

     For a Thin Application client application, you also need to specify the actual location of the `sas.client.props` and `ssl.client.props` files located in the Application Clients run-time dependency that is installed in the JWS cache.

     ```
     <argument>-CCDcom.ibm.CORBA.ConfigURL=file:${WAS_ROOT}/properties/sas.client.props </argument>
     <argument>-CCDcom.ibm.SSL.ConfigURL=file:${WAS_ROOT}/properties/ssl.client.props </argument>
     ```

     If any of the default settings in the `sas.client.props` and `ssl.client.props` file need modifying, use the –CCD to change the settings through the system properties, as shown in the following example:

     ```
     <argument>-CCDjavacom.ibm.CORBA.securityEnabled=false </argument>
     ```

     Note: The ${WAS_ROOT} token used in the JNLP file is replaced by the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, to the actual location of the Application Clients run-time dependency installation in the JWS cache. If you are using JSP to dynamically create this JNLP description file, you must escape this token because it has a different meaning in JSP 2.0. See the following example for details:

     ```
     <argument>-CCDcom.ibm.ssl.keyStore=\${WAS_ROOT}/etc/key.p12 </argument>
     <argument>-CCDcom.ibm.ssl.trustStore=\${WAS_ROOT}/etc/trust.p12 </argument>
     ```

JNLP descriptor file for a J2EE Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application. If it is a J2EE Application client application (that is the resources for the application contain an EAR file with a client application), then the launcher class starts a second Java Virtual Machine

(JVM) using the JRE provided by the Application Clients run-time dependency and launches the J2EE Application client application that is packaged in the EAR file.

Here is an example of the client application JNLP descriptor file for a J2EE Application client application.

```
<!--
"This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for
redistribution by customer, as part of such an application, in customer's
own products."

Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2004
All Rights Reserved * Licensed Materials - Property of IBM
-->
<!--
 This is a generic jnlp for a client app.  It will specify the WAS JRE
  as a dependency as well as the client launcher
-->
<!-- ========================================================================= -->
<!-- TODO: change "codebase" to the actual url location of this jnlp file -->
<!-- ========================================================================= -->
<jnlp spec="1.0+"
 codebase="http://your_server:port_number/J2EEWebStartWeb/JavaClients/WebStartExample">

  <information>
    <title>J2EE Client Exampletitle>J2EE Client Example</title>
    <vendor>IBM Client Teamvendor>IBM Client Team</vendor>
    <description>JVE WebStart example of: J2EE Client Example</description>
    <description kind="short">J2EE Client Example</description>
    <description kind="tooltip">J2EE Client Example</description>
  </information>

  <security>
    <all-permissions/>
  </security>

  <resources>

    <!-- ============================================================================ -->
    <!-- TODO: Update the "version" value to match your Application Client runtime -->
    <!--      Update the "href" value to the url for downloading the Application  -->
    <!--       Client runtime.                                                     -->
    <!-- ============================================================================ -->
    <j2se version="WASClient6.1.0"
     href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJre/AppClientRT.jsp"/

    <!-- The client app will require a client launcher -->
    <jar href="../Launcher/WebSphereClientLauncher.jar" main="true"/>

    <!-- Ear we want to download to the client -->
    <jar href="J2eeJWS.ear"/>

    <!-- The launcher depends on this property to be set -->
    <property name="com.ibm.websphere.client.launcher.ear" value="J2eeJWS.ear"/>

  </resources>

  <!-- WebStart will consider the Launcher as the app. to run  -->
  <application-desc main-class="com.ibm.websphere.client.launcher.ClientLauncher">
    <argument>-CCproviderURL=corbaloc:iiop:your_server</argument>
    <argument>-CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12</argument>
    <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12</argument>
  </application-desc>
</jnlp>
```

JNLP descriptor file for a Thin Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application. If it is a Thin Application client application, then the launcher class uses the current JVM from the Application Clients run-time dependency and invokes the Thin Application client application main method.

Here is an example of the JNLP descriptor file for a Thin Application client application.

```
<!--
"This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for
redistribution by customer, as part of such an application, in customer's
own products."

Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2004
All Rights Reserved * Licensed Materials - Property of IBM
-->

<!--
   This is a generic jnlp for a client app.  It will specify the WAS JRE
   as a dependency as well as the client launcher
 -->

<!-- ========================================================================= -->
<!-- TODO: change "codebase" to the actual url location of this jnlp file -->
<!-- ========================================================================= -->
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://your_server:port_number/J2EEWebStartWeb/JavaClients/WebStartExample">
  <information>
    <title>Thin Basic Calculator Client Samples</title>
    <vendor>IBM</vendor>
    <description>Thin Basic Calculator Client Samples</description>
    <offline-allowed/>
  </information>

  <security>
    <all-permissions/>
  <security>

  <resources>
    <j2se version="WASClient6.1.0"
     href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJre/AppClientRT.jspz"/>


    <extension name="WAS Thin EJB Client Library"
               href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/AppClientLib.jnlp"/>

   <!-- you must use the jar resource for JWS LaunchClient class here if using JWS LaunchClient
 wrapper launcher -->
   <jar href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/WebSphereClientLauncher.jar" main="true"/>

   <jar href="BasicCalculatorClientCommon.jar"/>
   <jar href="BasicCalculatorEJB.jar"/>
   <jar href="BasicCalculatorThinClient.jar"/>

   <property name="com.ibm.websphere.client.launcher.main"
value="com.ibm.websphere.samples.technologysamples.basiccalcthinclient.BasicCalculatorClientThinMain"/>
  <property name="java.naming.factory.initial"
            value="com.ibm.websphere.naming.WsnInitialContextFactory" />
  <property name="java.naming.provider.url"
            value="corbaloc:iiop:your_server:port_number"/>
  <property name="com.ibm.CORBA.ConfigURL"
```

```
value="http://your_server:port_number/J2EEWebStartWeb/JavaClients/sas.client.props"/>
  <property name="com.ibm.SSL.ConfigURL"

value="http://your_server:port_number/J2EEWebStartWeb/JavaClients/ssl.client.props"/>

  <!-- *** Logging Properties ***
       <property name="com.ibm.websphere.client.launcher.jws.trace" />
       <property name="java.util.logging.configureByServer" value="true" />
       <property name="traceSettingsFile" value="TraceSettings.properties" />
       <property name="com.ibm.CORBA.Debug" value="true" />
       <property name="com.ibm.CORBA.CommTrace" value="true" />
       <property name="java.util.logging.manager" value="com.ibm.ws.bootstrap.WsLogManager" />
       <property name="com.ibm.CORBA.RasManager" value="com.ibm.websphere.ras.WsOrbRasManager" />
  -->
  </resources>
    <application-desc>
      <argument>-CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12</argument>
      <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12</argument>
      <argument>add</argument>
      <argument>1</argument>
      <argument>2</argument>
  </application-desc>
</jnlp>
```

ClientLauncher class:

The class, com.ibm.websphere.client.installer.ClientLauncher, contains a main() method that is called by Java Web Start (JWS) to launch the client application. The Java Web Start client is used with platforms that support a Web browser.

This client is packaged in the `WebSphereClientLauncher.jar` file that is located in a WebSphere Application Server clients installation under the *<app_server_root>*/ JWS directory.

This launcher class configures the run-time environment for J2EE application clients and thin client applications (not J2EE application clients).

The launcher class requires that the following properties are defined. These properties are not defined in a separate properties file. Instead, the properties are defined as part of the Java Network Launching Protocol (JNLP) files.

com.ibm.websphere.client.launcher.main
: If the client application is a Thin Application client, then this property should be specified. It specifies the class where the main entry point of the client application resides.

com.ibm.websphere.client.launcher.ear
: If the client application is a J2EE Application client, then this property should be specified. It specifies the name of the EAR file to be executed. This property takes precedence over com.ibm.websphere.client.launcher.main. However, only one of the two properties should be specified.

Launcher tool:

The launcher configures the run-time environment for J2EE application clients and thin client applications (not J2EE application clients). The launcher utility is located in the main entry point of the Java Network Launching Protocol (JNLP) application client. The main class launcher name is com.ibm.websphere.client.launcher.ClientLauncher and is located in the `WebSphereClientLauncher.jar` file.

The launcher tool requires that the following properties are defined.

com.ibm.websphere.client.launcher.main
If the client that is to be run is a thin client, then this property should be specified. It specifies the class where the main entry point of the application resides.

com.ibm.websphere.client.launcher.ear
If the client that is to run is the J2EE client, then this property should be specified. It specifies the name of the ear file to be executed. This property takes precedence over com.ibm.websphere.client.launcher.main although only one of the two properties should be specified.

com.ibm.websphere.client.launcher.classpath.* (required for J2EE client applications only)
There can be a set of properties that are prefixed with com.ibm.websphere.client.launcher.classpath. Each property specifies a JAR file that is to be added to the class path of the application. This JAR file is a JAR file that is already defined as a resource for the application. This file is needed so that the correct elements of the class path of the Java Virtual Machine (JVM) starting the client launcher can be retrieved and added to the class path of the (JVM) that is to be spawned for the application client.

These properties are not defined in a separate properties file. Instead, they are defined as part of the Java Network Launching Protocol files.

Preparing the Application Client run-time dependency component for Java Web Start

For a J2EE application client application and or Thin application client application to be launched using Java Web Start (JWS), an Java Runtime Environment that IBM provides, the library JAR files and properties files bundled in Application Client for WebSphere Application Server must be installed in the JWS. This article provides the steps to build the Application Client run-time dependency component from an Application Client installation. It is packaged as a Web Archive Resource (WAR) file that can be installed in an Application Server.

Install the Application Client for WebSphere Application Server for the platform to which the client application deploys. If there is a requirement to deploy the client application to multiple platforms, the Application Client run-time dependency component must be built separately for each platform that client application supports.

For example, if the client application deploys to both the Windows platform and Linux platform, follows the steps for this task to build the Application Client run-time dependency component for Windows on a Windows platform machine with the Application Client for WebSphere Application Server for Windows installed. Now, repeat the steps for this task to build the Application Client run-time dependency component for Linux on a Linux platform machine with the Application Client for WebSphere Application Server for Linux installed.

1. Install the Application Client for WebSphere Application Server for the client application supported operating systems. Install Application Client in the `C:\Program Files\IBM\WebSphere\AppClient` directory.
2. Change the directory to the installation bin directory. See the following example for help:

   ```
   CD C:\Program files\IBM\WebSphere\AppClient\bin
   ```
3. Run the buildClientRuntime tool to generate the Application Client run-time JAR file in a temporary directory which contains the Java 2 Runtime Environment, Application Client run-time properties, the SSL KeyStore and TrustStore file, and the Application Client run-time library JAR files. See the following example for help:

   ```
   buildClientRuntime C:\WebApp1\runtime\WASClient6.1_windows.jar
   ```

   If you are building an Application Client run-time JAR file only for serving Thin application client applications and not for J2EE application client applications, you can reduce the size of the generated JAR file by not including the Application Client run-time library JAR files. An extra parameter is passed to the buildClientRuntime tool, as the following example shows:

```
buildClientRuntime C:\WebApp1\runtime\WASClient6.1_windows.jar
                   buildThin
```

4. Copy the WebSphereClientRuntimeInstaller.jar file to the same location of the JAR file generated in the previous step. This JAR file is located in the JWS directory of the WebSphere Application Server clients installation. See the following example for help:

   ```
   copy ..\JWS\WebSphereClientRuntimeInstaller.jar   C:\WebApp1\runtime
   ```

5. Sign the JAR files created from the previous steps, using the Java 2 SDK jarsigner utility, as the following example shows:

   ```
   cd C:\WebApp1\runtime

   jarsigner -keystore myKeystore -storepass myPassword
                 WASClient6.1_windows.jar myKeyAliasName

   jarsigner -keystore myKeystore -storepass myPassword
                 WebSphereClientRuntimeInstaller.jar myKeyAliasName
   ```

   a. This step also requires you to create a keystore file, such as myKeystore.

   b. You must also create a self-signed certificate for the myKeystore file.

   Note: When running the JAR signer tool on HP platforms, add the -J″-Xmx256m″ flag to the jarsigner command to increase the available heap size and prevent the error, OutOfMemoryError. See the following example for help:

   ```
   jarsigner -J"-Xmx256m" -keystore myKeystore -storepass myPassword
                   WebSphereClientRuntimeInstaller.jar myKeyAliasName
   ```

6. Create an Application Client run-time installer JNLP descriptor file or a JavaServer Pages (JSP) file if it is generated dynamically in the same temporary directory as previous step. See the sample JNLP file shown in the Example section of this topic.
7. Package the two signed JAR files and the Application Client run-time installer JNLP descriptor file into a WAR file. This WAR file is packaged into an EAR file that can be deployed to an Application Server.

Your Web application is ready to serve the Application Client run time and the JRE environment.

```
<!-- This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer (a) for its own instruction and study, (b) in order
to develop applications designed to run with an IBM WebSphere product, either for customer's
own internal use or for redistribution by customer, as part of such an application, in
customer's own products.

Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2005
All Rights Reserved * Licensed Materials - Property of IBM
-->

<%-- // to set the Last_Modified header so that the JNLP client will know whether to download
     // the JNLP file again and update the cached copy.
     String jspPath = application.getRealPath(request.getServletPath());
     java.io.File jspFile = new java.io.File(jspPath);
     long lastModified = jspFile.lastModified();
%><%
      // locally declared variables
      String url=request.getRequestURL().toString();
      String jnlpCodeBase=url.substring(0,url.lastIndexOf('/'));
      String jnlpRefURL=url.substring(url.lastIndexOf('/')+1,url.length());

      // Need to set a JNLP mime type - if WebStart is installed on the client,
      // this header will induce the browser to drive the WebStart Client
      response.setContentType("application/x-java-jnlp-file");                          1
      response.setHeader("Cache-Control", null);
      response.setHeader("Set-Cookie", null);
      response.setHeader("Vary", null);
      response.setDateHeader("Last-Modified", lastModified);

      // An installer must reply with the version number for a given install
```

```
      if (response.containsHeader("x-java-jnlp-version-id"))
        response.setHeader("x-java-jnlp-version-id", "WASClient6.1.0");               2
      else
        response.addHeader("x-java-jnlp-version-id", "WASClient6.1.0");
%>

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

<!-- ========================================================== -->
<!-- TODO: change "codebase" to the actual url location of this jsp -->
<!-- ========================================================== -->

<jnlp spec="1.0+"
codebase="http://YOUR_APP_SERVER:PORTNUMBER/WEBAPP_CONTEXT_ROOT/Runtime/WebSphereJre">

<information>
  <title>Application Client Java Runtime Environment</title>
  <vendor>IBM</vendor>
  <icon href="icon.gif"/>
  <description>Application Client Java Runtime Environment</description>
  <description kind="short">Applicaiton Client JRE</description>
  <description kind="tooltip">Applicaiton Client JRE</description>
  <offline-allowed/>
</information>

<security>
 <all-permissions/>
</security>

 <resources>
  <j2se version="1.4+"/><%-- The installer can use any 1.4 JRE --%> 3
   <jar href="WebSphereClientRuntimeInstaller.jar" main="true"/> 4

  <!-- JRE version registration with Web Start -->
  <property name="com.ibm.websphere.client.jre.version" value="WASClient6.1.0"/> 5
 </resources>

 <resources os="Windows"> 6
<!-- ========================================================== -->
<!-- TODO: the property value for unix platform is "java/jre/bin/javaw" -->
<!--   and the "os" value match to your target client machine platform             -->
<!-- ========================================================== -->

   <jar href="WASClient6.1.0_Windows.jar"/> 7

<!-- ========================================================== -->
<!-- TODO:  property value for unix platform is "java/jre/bin/javaw" -->
<!-- ========================================================== -->
<!-- relative path of the jre executable —->

   <property name="com.ibm.websphere.client.jre.launch.java"
 value="java\jre\bin\javaw.exe"/> 8

 </resources>
 <installer-desc main-class="com.ibm.websphere.client.installer.ClientRuntimeInstaller"/>
</jnlp>
```

1. Specifies that the file is a JNLP mime type so that the browser can process the JNLP file.
2. Specifies the exact version of this Application Client run-time dependency component in the response by setting the HTTP header field: `x-java-jnlp-version-id`.
3. Specifies the required JRE version to run the installer program.
4. Specifies the installer `WebSphereClientRuntimeInstaller.jar` file, which contains the ClientRuntimeInstaller class.
5. Specifies a system property that defines the version of Application Client run-time dependency component. This version is registered to the JNLP client.

6. Specifies resources for a particular platform. Each supported client application platform needs its own separate JAR file.
7. Specifies the Application Client run-time dependency component JAR file.
8. Specifies the program to call that starts a JVM for the client application.

Preparing Application Client run-time library component for Java Web Start.

buildClientRuntime tool:

For a J2EE application client application and or Thin application client application to be launched using Java Web Start (JWS), the library JAR files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to build those JAR files. The Java Web Start client is used with platforms that support a Web browser. .

The buildClientRuntime tool builds the required components from the WebSphere Application Server clients installation into the JAR file specified on the command. This JAR file contains:

- License files
- Java 2 Runtime Environment (JRE) that IBM provides
- Application Clients run-time properties and configuration
- SSL KeyStore and TrustStore files
- Run-time library JAR files

In the case of building an Application Clients run-time JAR file only for serving Thin Application client applications and not for J2EE Application client applications, the run-time library JAR files and the Application Clients run-time properties files are not included, except the configuration files, `sas.client.props`, `ssl.client.props` and `soap.client.props`, located in the `WAS_ROOT/properties` directory. The Java Web Start client is used with platforms that support a Web browser.

The command-line invocation syntax for the buildClientRuntime tool is shown in the following example:

```
Windows Usage:    buildClientRuntime .bat  jar_file  [type]
Unix Usage:         buildClientRuntime.sh    jar_file  [type]
Where:
        jar_file
Specifies the target jar file name.

Range:
                    buildJ2EE - Default value that builds a Application Clients
                                run-time library for J2EE application.
                    buildThin - Builds a Application Clients run-time library
                                for Thin application.
```

ClientRuntimeInstaller class: This class, com.ibm.websphere.client.installer.ClientRuntimeInstaller, contains a main() method that Java Web Start (JWS) calls to install the Application Client for WebSphere Application Server run-time dependency component in JWS cache. It is packaged in `WebSphereClientRuntimeInstaller.jar` file located in the Application Client for WebSphere Application Server installation in the *<app_server_root>*/JWS directory.

Specify the `WebSphereClientRuntimeInstaller.jar` file and the Application Client run-time dependency component JAR file as JAR resources in the Application Client run-time installer Java Network Launcher Protocol (JNLP) descriptor file. See the following example for details:

```
<jar href="Launcher/WebSphereClientRuntimeInstall.jar" main="true"/>
<jar href="Launcher/WASClient6.1_windows.jarRuntimeInstall.jar" main="true"/>
```

The ClientRuntimeInstaller class main method requires the following properties to be set in the JNLP file:

com.ibm.websphere.client.jre.version
Specifies a Java Runtime Environment (JRE) version name that is to be used when referring to the Application Client run-time dependency component.

com.ibm.websphere.client.jre.launch.java
Specifies the relative location of the javaw.exe program in the Application Client run-time dependency component JAR file.

The previously mentioned properties, JRE version name and the location of the javaw.exe program are registered to the Java Web Start Application Manager, as shown in the following example:

```
<property name="com.ibm.websphere.client.jre.version" value="java\jre\bin\javaw.exe"/>
<property name="com.ibm.websphere.client.jre.launch.java" value="WASclient6.1"/>
```

Preparing Application Clients run-time library component for Java Web Start

The Java Web Start client is used with platforms that support a Web browser. For a Thin Application client application to be launched using Java Web Start (JWS), you also need to create a Java Network Launching Protocol (JNLP) component to serve the Application Clients run-time library JAR files from the Application server. This JNLP component is referenced in the client application JNLP file with the <extension> tag. This article provides the steps to build the Application Clients run-time library component from an Application Clients installation. It is packaged as its own Web Archive Resource (WAR) file or to the same WAR file that contains the Application Clients run-time dependency component, and can be installed in an Application server.

Install the Application Client for WebSphere Application Server for the platform to which client applications deploy.

1. Install the Application Clients on the client application supported operating system. For example, install Application Clients in the `C:\Program Files\IBM\WebSphere\AppClient` directory.
2. Change the directory to the installation bin directory. For example:

   ```
   CD C:\Program files\IBM\WebSphere\AppClient\bin
   ```
3. Run `buildClientLibJars` to copy the Application Clients run-time library JAR files from the Application Clients installation to a temporary directory. All the JAR files in the temporary directory are signed, as shown in the following example:

   ```
   buildClientLibJars C:\WebApp1\runtime\WebSphereJars
                      myKeystore myPassword myKeyAliasName
   ```
   a. This step also requires you to create a keystroke file, such as `myKeystore`.
   b. You must also create a self-signed certificate for the `myKeystore` file.
4. Create an Application Clients run-time installer JNLP descriptor file in the same temporary directory as the previous step. See the sample JNLP file shown in the Example section of this topic.
5. Package these JAR files and the Application Clients run-time library component JNLP descriptor file into a WAR file. You can also package both Application Clients run-time library component and Application Clients run-time dependency component in the same WAR file. This WAR file is packaged into an EAR file that can deployed to an Application server.

```
<!--
  "This sample program is provided AS IS and can be used, executed, copied
and modified without royalty payment by customer (a) for its own instruction
and study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for redistribution
by customer, as part of such an application, in customer's own products."
Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2005
All Rights Reserved * Licensed Materials - Property of IBM
-->
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://YOUR_APP_SERVER:PORTNUMBER/WEBAPP_CONTEXT_ROOT/Runtime/WebSphereJars">
 <information>
  <title>Application Client Library</title>
  <vendor>IBM</vendor>
```

```
  <icon href="icon.gif"/>
  <description>Application Client Library</description>
  <description kind="short">Application Client Library</description>
  <description kind="tooltip">Application Client Library</description>
  <offline-allowed/>
 </information>

<security>
 <all-permissions/>
</security>

<component-desc/>
 <resources><jar href="activation-impl.jar"/>
<jar href="bootstrap.jar"/>
<jar href="com.ibm.events.client_6.1.0.jar"/>
<jar href="com.ibm.mq.jar"/>
<jar href="com.ibm.mqjms.jar"/>
<jar href="com.ibm.uddi.client_1.0.0.jar"/>
<jar href="com.ibm.ws.bootstrap_6.1.0.jar"/>
<jar href="com.ibm.ws.debug.osgi_6.1.0.jar"/>
<jar href="com.ibm.ws.emf_2.0.0.jar"/>
<jar href="com.ibm.ws.j2ee.client_6.1.0.jar"/>
<jar href="com.ibm.ws.runtime.dist_6.1.0.jar"/>
<jar href="com.ibm.ws.runtime.gateway_6.1.0.jar"/>
<jar href="com.ibm.ws.runtime_6.1.0.jar"/>
<jar href="com.ibm.ws.security.crypto_6.1.0.jar"/>
<jar href="com.ibm.ws.sib.client_2.0.0.jar"/>
<jar href="com.ibm.ws.sib.utils_2.0.0.jar"/>
<jar href="com.ibm.ws.wccm_6.1.0.jar"/>
<jar href="com.ibm.wsspi.extension_6.1.0.jar"/>
<jar href="dhbcore.jar"/>
<jar href="j2ee.jar"/>
<jar href="launchclient.jar"/>
<jar href="lmproxy.jar"/>
<jar href="mail-impl.jar"/>
<jar href="org.eclipse.core.runtime_3.1.1.jar"/>
<jar href="org.eclipse.osgi_3.1.1.jar"/>
<jar href="org.eclipse.update.configurator_3.1.0.jar"/>
<jar href="properties.jar"/>
<jar href="serviceadapter.jar"/>
<jar href="startup.jar"/>
<jar href="urlprotocols.jar"/>
<jar href="WebSphereClientLauncher.jar"/>
 <resources/><jnlp/>
```

buildClientLibJars tool:

For a J2EE application client application and or Thin application client application to be launched using Java Web Start (JWS), the properties files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to create those property JAR files. The Java Web Start client is used with platforms that support a Web browser.

The buildClientLibJars tool copies the JAR files from the Application Client for WebSphere Application Server installation and creates a `properties.jar` file, which contains the properties files from the Application Clients installation properties directory to a specified location. When this property is created, the tool uses the value of `keystore`, `storepass` and `alias` to sign all the JAR files in the specified location.

```
Windows Usage: buildClientLibJars.bat  target_dir keystore  storepass  alias
Unix Usage:        buildClientLibJars.sh  target_dir keystore  storepass  alias
Where:
     target_dir       Specifies the target directory where the Application
                           Clients library JAR files copied to.
    keystore         Specifies a keystore file.
    storepass       Specifies the keystore password.
    alias               Specifies an alias for the key object in the key file.
```

Using the Java Web Start sample

The EAR file, WebSphereClientRuntime.ear, is provided in the JWS directory of the Client Application for WebSphere Application Server installation. This EAR file provides a sample Application Clients run-time installer JNLP descriptor file and a sample Application Clients run-time library component JNLP descriptor file. Follow the steps in this task to build the Application Clients run-time dependency component and the Application Clients run-time library component. Add these components to the WebSphereClientRuntime.ear file, and then install the EAR file in an Application Server to be used by the client application.

Install the Application Client for WebSphere Application Server for the platform to which the client application deploys. If there is a requirement to deploy the client application to multiple platforms, the Application Clients run-time dependency component must be built separately for each platform that the client application supports.

1. Install the Application Clients on the client application supported operating system. For example, install Application Clients in the C:\Program Files\IBM\WebSphere\AppClient directory.
2. Create the following temporary working directories:

   ```
   MKDIR C:\WebApp1
   MKDIR C:\WebApp1\runtime
   MKDIR C:\WebApp1\runtime\Widnows
   MKDIR C:\WebApp1\runtime\WebSphereJars
   ```

3. Change directory to the installation bin directory. See the following example for help:

   ```
   CD C:\Program files\IBM\WebSphere\AppClient\bin
   ```

4. Run the buildClientRuntime tool to generate the Application Clients run-time JAR file in a temporary directory that contains the Java 2 Runtime Environment that IBM provides, Application Clients run-time properties, the SSL KeyStore and TrustStore files, and the Application Clients run-time library JAR files. See the following example for details:

   ```
   buildClientRuntime C:\WebApp1\runtime\windows\WASClient6.1.0_Windows.jar
   ```

5. Copy the WebSphereClientRuntimeInstaller.jar file to the same location of the JAR file generated in the previous step. This JAR file is located in the JWS directory of the Application Client for WebSphere Application Server installation. For example, copy the ..\JWS\WebSphereClientRuntimeInstaller.jar file to the C:\WebApp1\runtime directory.
6. Sign the JAR files created from the previous steps, using the Java 2 SDK jarsigner utility. See the following example for details:

   ```
   cd C:\WebApp1\runtime

   jarsigner -keystore myKeystore -storepass myPassword
                   WASClient6.1_windows.jar myKeyAliasName

   jarsigner -keystore myKeystore -storepass myPassword
                   WebSphereClientRuntimeInstaller.jar myKeyAliasName
   ```

   a. This step also requires you to create a keystore file, such as myKeystore.
   b. You must also create a self-signed certificate for the myKeystore file.
7. Run buildClientLibJars to copy the Application Clients run-time library JAR files from the Application Client for WebSphere Application Server installation to a temporary directory. All the JAR files in the temporary directory are signed. See the following example for details:

   ```
   buildClientLibJars C:\WebApp1\runtime\WebSphereJars
                      myKeystore myPassword myKeyAliasName
   ```

   a. This step also requires you to create a keystroke file, such as myKeystore.
   b. You must also create a self-signed certificate for the myKeystore file.
8. Add all the JAR files created in the previous steps in the C:\WebApp1 directory to the WAR file within the WebSphereClientRuntime.ear file. The contents of the WAR file are shown in the following example:

   ```
   The root of the WAR
   ├──META-INF
   │                MANIFEST.MF
   │
   ```

```
├──Runtime
│    ├──WebSpherejars
│    │         AppClientLib.jnlp
│    │         com.ibm.ws.runtime_6.1.0.jar
│    │         com.ibm.ws.j2ee.client_6.1.0.jar
│    │         com.ibm.ws.wccm_6.1.0.jar
│    │                  :
│    │         (all the jars created in step 7 under
│    │         c:\WebApp1\Runtime\WebSphereJars)
│    └──WebSphereJre
│           AppClientRT.jsp
│           WASClient6.1.0_Windows.jar
│           WebSphereClientLauncher.jar
│           WebSphereClientRuntimeInstaller.jar |
│
└──WEB-INF
       ibm-web-bnd.xmi
       ibm-web-ext.xmi
       web.xml
```

9. Install the `WebSphereClientRuntime.ear` file to an Application Server. You have just created an Application Clients run-time dependency component and Application Clients run-time libraries for serving J2EE Application client applications and Thin Application client applications using Java Network Launching Protocol (JNLP) or Java Web Start (JWS).

Installing Java Web Start

This topic provides the steps necessary to install JWS on the AIX platform.

Before you begin this task, see the following topics to understand Java Web Start (JWS) technology and its components:

- Prepare the Application Clients run-time dependency component for JWS
- Prepare the Application Clients run-time library component for JWS

Note: You can use Java Web Start on Java 2 Standard Edition Developer Kits that IBM provides, packaged in the Application Client for WebSphere Application Server, Version 6; Java Web Start on Sun Microsystems J2SE Software Development Kit or J2SE Java Runtime Environment 1.4.2, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems, or the Java Web Start on HP SDK or RTE for Java 2 version 1.4.2, which you can download from the HP Web site.

Use the following steps to install JWS on the AIX platform.

1. Install IBM Application Client for WebSphere Application Server.
2. Change your directory to the `client_install_root/java/jre` path.
3. Run `sh bin/webstart_install_sdk.sh`.
4. When prompted for the Java path, enter your JRE path. Use the following example: `client_install_root/java/jre`.
5. You should see the following messages, which indicate that JWS installs successfully:
   - Obtaining version...
   - You appear to be running 1.4.2
   - Extracting...
   - Updating ~/.mailcap...
   - Updating ~/.mime.types...
6. Change your path to the JWS installed path. For example, enter `client_install_root/java/jre/javaws/`.
7. Run `./javaws` from the path mentioned in the previous step.

Java Web Start for Application client best practices

Do not use JSP to dynamically generate a JNLP file, otherwise the JNLP jsp page cannot be opened in some IE browsers. To use a static JNLP file, you will need to add the following mime type mapping in the web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
       <display-name>
       WAS Client runtime for Java Web Start</display-name>
       <welcome-file-list>
              <welcome-file>index.html</welcome-file>
              <welcome-file>index.htm</welcome-file>
              <welcome-file>index.jsp</welcome-file>
              <welcome-file>default.html</welcome-file>
              <welcome-file>default.htm</welcome-file>
              <welcome-file>default.jsp</welcome-file>
       </welcome-file-list>
       <mime-mapping>
       <extension>jnlp</extension>
       <mime-type>application/x-java-jnlp-file</mime-type>
       </mime-mapping>
</web-app>
```

Running application clients

The J2EE specification requires support for a client container that runs stand-alone Java applications (known as J2EE application clients) and provides J2EE services to the applications. J2EE services include naming, security, and resource connections.

You are ready to run your application client using this tool after you have:
1. Written the application client program.
2. Assembled and installed an application module (`.ear` file) in the application server run time.
3. Deployed the application using the Application Client Resource Configuration Tool (ACRCT) on Windows .

This task only applies to J2EE application clients. To launch J2EE application clients using the launchClient script, perform the following steps:

1. On the CL command line, enter the following command to start the Qshell environment:

   ```
   STRQSH
   ```

   a. Enter the following command to launch J2EE application clients:

      app_server_root`/bin/launchClient`
2. Pass parameters to the `launchClient` command or to your application client program as well. The `launchClient` command allows you to do both. The `launchClient` command requires that the first parameter is either:
   - An EAR file specifying the application client to launch.
   - A request for `launchClient` usage information.

   The following example illustrates the command line invocation syntax for the launchClient tool:

   ```
   launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]
   ```

   where
   - *userapp.ear* is the path and the name of the EAR file that contains the application client.
   - *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.

- *app args* are arguments that pass to the application client.
- *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation. The default is **default_profile**.
- *-JVMOptions* is a valid Java standard or non-standard option string. Insert quotation marks around the string.
- *-help, -?* prints the usage information.

All other parameters intended for the `launchClient` command must begin with the -CC prefix.

Parameters that are not EAR files, or usage requests, or that do not begin with the -CC prefix, are ignored by the application client run time, and are passed directly to the application client program.

The `launchClient` command retrieves parameters from three places:
- The command line
- A properties file
- System properties

The parameters are resolved in the order listed above, with command line values having the highest priority and system properties the lowest. Using this prioritization you can set and override default values.

3. Specify the server name.

   By default, the **launchClient** command uses the localhost for the `BootstrapHost` property value.

   This setting is effective for testing your application client when it is installed on the same computer as the server. However, in other cases override this value with the name of your server. You can override the `BootstrapHost` value by invoking `launchClient` command with the following parameters:

   ```
   launchClient myapp.ear -CCBootstrapHost=abc.midwest.mycompany.com
   ```

   You can also override the default by specifying the value in a properties file and passing the file name to the launchClient shell.

   Security is controlled by the server. You do not need to configure security on the client because the client assumes that security is enabled. If server security is not enabled, then the server ignores the security request, and the application client functions as expected.

You can store launchClient values in a properties file, which is a good method for distributing default values. You can then override one or more values on the command line. The format of the file is one `launchClient -CC` parameter per line without the `-CC` prefix. For example:

```
 verbose=true classpath=c:\mydir\util.jar;c:\mydir\harness.jar;c:\production\G19
\global.jar BootstrapHost=abc.westcoast.mycompany.com tracefile=c:\WebSphere\mylog.txt
```

launchClient tool

This topic describes the Java 2 Platform Enterprise Edition (J2EE) command line syntax for the launchClient tool for WebSphere Application Server.

The following example illustrates the command line invocation syntax for the launchClient tool:

```
launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]
```

where
- *userapp.ear* is the path and the name of the EAR file that contains the application client.
- *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.
- *app args* are arguments that pass to the application client.
- *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation.

The default is **default_profile**.

- *-JVMOptions* is a valid Java standard or nonstandard option string. Insert quotation marks around the string.
- *-help, -?* prints the usage information.

The first parameter must be `-help`, `-?` or contain no parameter at all. The `-profileName pName` and `-JVMOptions` options are optional parameters. If used, they must appear before the *<userapp>* parameter. All other parameters are optional and can appear in any order after the *<userapp>* parameter. The J2EE Application client run time ignores any optional parameters that do not begin with a `-CC` prefix and passes those parameters to the application client.

Client container parameters

Supported arguments include:

-CCadminConnectorHost
Specifies the host name of the server from which configuration information is retrieved.

The default is the value of the `-CCBootstrapHost` parameter or the value, `localhost`, if the `-CCBootstrapHost` parameter is not specified.

-CCadminConnectorPort
Indicates the port number for the administrative client function to use. The default value is `8880` for SOAP connections and `2809` for Remote Method Invocation (RMI) connections.

-CCadminConnectorType
Specifies how the administrative client connects to the server. Specify `RMI` to use the RMI connection type, or specify `SOAP` to use the SOAP connection type. The default value is `SOAP`.

-CCadminConnectorUser
Administrative clients use this user name when a server requires authentication. If the connection type is SOAP, and security is enabled on the server, this parameter is required. The SOAP connector does not prompt for authentication.

-CCadminConnectorPassword
The password for the user name that the `-CCadminConnectorUser` parameter specifies.

-CCaltDD
The name of an alternate deployment descriptor file. This parameter is used with the `-CCjar` parameter to specify the deployment descriptor to use. Use this argument when a client JAR file is configured with more than one deployment descriptor. Set the value to `null` to use the client JAR file standard deployment descriptor.

-CCBootstrapHost
The name of the host server you want to connect to initially. The format is:
your_server_of_choice.com

-CCBootstrapPort
The server port number. If you do not specify this argument, the WebSphere Application Server default value is `used`.

-CCclassLoaderMode
Specifies the class loader mode. If PARENT_LAST is specified, the class loader loads classes from the local class path before delegating the class loading to its parent. The classes loaded for the following are affected:

- Classes defined for the J2EE application client
- Resources defined in the J2EE application
- Classes specified on the manifest of the J2EE client JAR file
- Classes specified using the -CCclasspath option

If PARENT_LAST is not specified, then the default mode, PARENT_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.

-CCclasspath
A class path value. When you launch an application, the system class path is used. If you want to access classes that are not in the EAR file or part of the system class paths, specify the appropriate class path here. Multiple paths can be concatenated.

-CCD
Use this option to have the WebSphere Application Server set the specified system property during initialization. Do not use the equals (=) character after the `-CCD`. For example: `-CCDcom.ibm.test.property=testvalue`. You can specify multiple `-CCD` parameters. The general format of this parameter is `-CCD<property key>=<property value>`. For example, `-CCDI18NService.enable=true`.

-CCdumpJavaNameSpace
Prints out the Java portion of the Java Naming and Directory Interface (JNDI) name space for WebSphere Application Server. The `true` value uses the short format that prints out the binding name and the type of the object bound at that location. The `long` value uses the long format that prints out the binding name, bound object type, local object, type and string representation of the local object, for example, IORs and string values. The default value is `false`.

-CCexitVM
Use this option to have the WebSphere Application Server call the `System.exit()` method after the client application completes. The default is `false`.

-CCinitonly
Use this option to initialize application client run time for ActiveX application clients without launching the client application. The default is `false`.

-CCjar
The name of the client Java Archive (JAR) file that resides within the EAR file for the application you wish to launch. Use this argument when you have multiple client JAR files in the EAR file.

-CCpropfile
Indicates the name of a properties file that contains launchClient properties. Specify the properties without the `-CC` prefix in the file, with the exception of the securityManager, securityMgrClass and securityMgrPolicy properties. See the following example: `verbose=true`.

-CCproviderURL
Provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a Common Object Request Broker Architecture (CORBA) object URL or an Internet Inter-ORB Protocol (IIOP) URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. You can specify bootstrap server addresses, for all servers in the cluster, in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the `-CCBootstrapHost` and `-CCBootstrapPort` parameters. A CORBA object URL specifying multiple systems is illustrated in the following example:

```
-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809
```

This value is mapped to the `java.naming.provider.url` system property.

-CCsecurityManager
Enables and runs the WebSphere Application Server with a security manager. The default is `disable`.

-CCsecurityMgrClass
Indicates the fully qualified name of a class that implements a security manager. Only use this argument if the `-CCsecurityManager` parameter is set to `enable`. The default is `java.lang.SecurityManager`.

-CCsecurityMgrPolicy
Indicates the name of a security manager policy file. Only use this argument if the `-CCsecurityManager` parameter is set to enable. When you enable this parameter, the `java.security.policy` system property is set. The default is *<app_server_root>*/ `properties/client.policy`.

-CCsoapConnectorPort
The Simple Object Access Protocol (SOAP) connector port. If you do not specify this argument, the WebSphere Application Server default value is used.

-CCtrace
Use this option to obtain debug trace information. You might need this information when reporting a problem to IBM customer support. The default is `false`. For more information, read the topic ″Enabling trace.″

-CCtracefile
Indicates the name of the file to which trace information is written. The default is to write output to the console.

-CCtraceMode
Specifies the trace format to use for tracing. If the valid value, `basic`, is not specified the default is advanced. Basic tracing format is a more compact form of tracing.

For more information on basic and advanced trace formatting, see Interpreting trace output.

-CCverbose
This option displays additional information messages. The default is `false`.

The following examples demonstrate correct syntax.

Windows

```
launchClient c:\earfiles\myapp.ear -CCBootstrapHost=myWASServer -CCverbose=true app_parm1 app_parm2
```

Specifying the directory for an expanded EAR file

You can archive the `Manifest.mf` client Java Archive (JAR) files instead of automatically cleaning them up after the application exits.

Each time the launchClient tool is called, it extracts the Enterprise Archive (EAR) file to a random directory name in the temporary directory on your hard drive. Then the tool sets up the thread ClassLoader to use the extracted EAR file directory and JAR files included in the `Manifest.mf` client Java Archive (JAR) file. In a normal J2EE Java client, these files are automatically cleaned up after the application exits. This cleanup occurs when the client container shutdown hook is called. To avoid extracting the EAR file (and removing the temporary directory) each time the launchClient tool is called, complete the following steps:

1. Specify a directory to extract the EAR file by setting the `com.ibm.websphere.client.applicationclient.archivedir` Java system property. If the directory does not exist or is empty, the EAR file is extracted normally. If the EAR file was previously extracted, the launchClient tool reuses the directory.
2. Delete the directory before running the launchClient tool again, if you need to update your EAR file. When you call the `launchClient` command, it extracts the new EAR file to the directory. If you do not delete the directory or change the system property value to point to a different directory, the launchClient tool reuses the currently extracted EAR file and does not use your changed EAR file. When specifying the `com.ibm.websphere.client.applicationclient.archivedir` property, make sure that the directory you specify is unique for each EAR file you use. For example, do not point the `MyEar1.ear` and the `MyEar2.ear` files to the same directory.

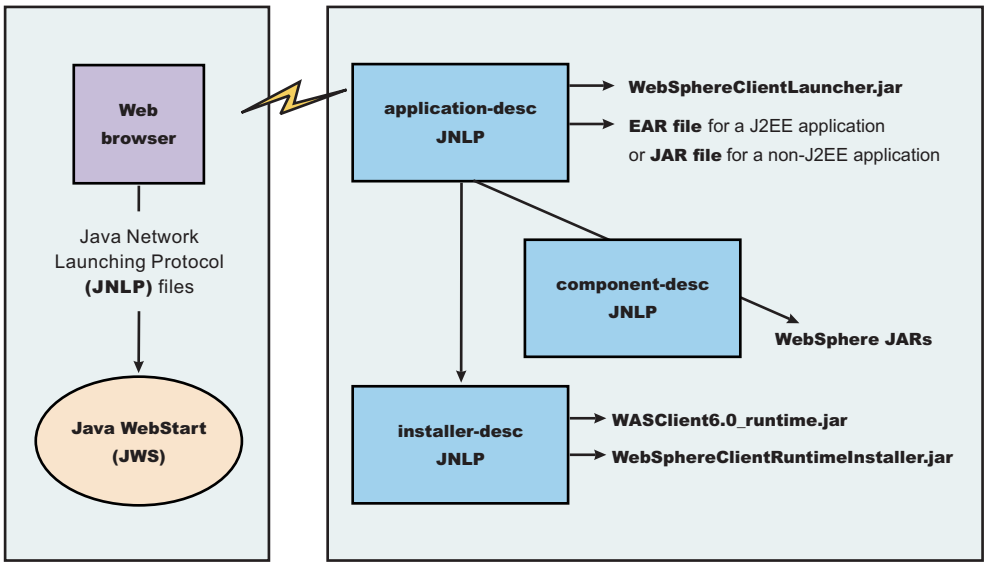
Java Web Start architecture for deploying application clients

Java Web Start is an application-deployment technology that includes the portability of applets, the maintainability of servlets and JavaServer Pages (JSP) file technology, and the simplicity of mark-up languages such as XML and HTML. It is a Java application that allows full-featured Java 2 client applications to be launched, deployed and updated from a standard Web server. The Java Web Start client is used with platforms that support a Web browser.

Upon launching Java Web Start for the first time, you might download new client applications from the Web. Each time you launch JWS thereafter, you can initiate applications either through a link on a Web page or (in Windows) from desktop icons or the Start menu. You can deploy applications quickly using Java Web Start, cache applications on the client machine, and launch applications remotely offline. Additionally, because Java Web Start is built from the J2EE infrastructure, the technology inherits the complete security architecture of the J2EE platform.

The technology underlying Java Web Start is the Java Network Launching Protocol & API (JNLP). Java Web Start is a JNLP client and it reads and parses a JNLP descriptor file (JNLP file). Base on the JNLP descriptor, it downloads appropriate pieces of a client application and any of its dependencies. If any of the pieces of the application are already cached on the client machine, then those components are not downloaded again, unless they have been updated on the server machine. After you download and cache the client application, JWS launches it natively on the client machine.

The following diagram shows an overview of launching a client application, include the Application Client for WebSphere Application Server, Version 6 as a dependent resource, using Java Web Start.



The Web browser running on a client machine connects to a Web application located on a server machine. The client application JNLP descriptor file is downloaded and processed by Java Web Start on the client machine.

In this diagram, there are three JNLP descriptor files:

- Client application JNLP descriptor (application-desc in the diagram)
- Application Clients run-time installer JNLP descriptor (installer-desc in the diagram)
- Application Clients run-time library component JNLP descriptor (component-desc in the diagram)

Each of these JNLP descriptor files, the client application (JAR or EAR) and the dependent resource JAR files are packaged as Web applications in an EAR file. This EAR file is deployed to an Application server. The client machine with JWS installed uses a Web browser to connect to the url of the client application JNLP descriptor file to download and run the client application.

Using Java Web Start from J2SE Java Runtime Environment 5.0 or later is highly recommended. All the platforms supported by the application client for WebSphere Application Server are supported with the exception Linux on Power and OS400 platforms.

You can use any of the following:

- Java Web Start on the Java 2 Standard Edition Developer Kits that IBM provides, packaged in Application Client for WebSphere Application Server, Version 6.1
- Java Web Start on Sun Microsystems J2SE Software Development Kit or J2SE Java Runtime Environment 5.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems
- Java Web Start on HP-UX JDK or JRE for Java 2 Platform Standard Edition, version 5, which you can download from the HP Web site

Using Java Web Start

This topic provides the steps and prerequisites necessary to use Java Web start.

Before you begin this task, see the following topics to understand Java Web Start technology and its components:

- “Java Web Start architecture for deploying application clients” on page 247
- “Client application Java Network Launcher Protocol deployment descriptor file” on page 250
- “ClientLauncher class” on page 254

Note: You can use any of the following:

- Java Web Start on Java 2 Standard Edition Developer Kits that IBM provides, packaged in the Application Client for WebSphere Application Server, Version 6.1
- Java Web Start on Sun Microsystems J2SE Software Development Kit or J2SE Java Runtime Environment 5.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems
- Java Web Start on HP-UX JDK or JRE for Java 2 Platform Standard Edition, version 5.0, which you can download from the HP Web site.

1. Prepare the Application Clients run-time dependency component for JWS.
2. Prepare the Application Clients run-time library component for JWS.
3. Installing JWS.
4. **Optional:** Run the Java Web Start sample.

   Problem: When you run Web services clients from Java Web Start using a Mozilla browser, you might get errors if the client argument contains quotations in the jnlp.jsp file. For example, the following argument results in an error:

   ```
   <argument>-url="wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
   /pli/ejb/WSMultiProtocolHome&"</argument>
   ```

   Error: The following errors display in the Java Web Start console:

   If using the EJB protocol, the following error is displayed:

   ```
   Client caught exception getting the InsuranceWebServicesPort
   using the URL
   "wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
   /pli/ejb/WSMultiProtocolHome&"
   java.net.MalformedURLException: no protocol:
   ```

```
"wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
/pli/ejb/WSMultiProtocolHome&"
 at java.net.URL.<init>(URL.java(Compiled Code))
 at java.net.URL.<init>(URL.java(Compiled Code))
 at java.net.URL.<init>(URL.java:411)
 at com.ibm.wssvt.tc.pli.webservice.InsuranceWebServicesClient
.getInsuranceServicesClientURL(InsuranceWebServicesClient.java:231)
 at com.ibm.wssvt.tc.pli.webservice.InsuranceWebServicesClient
.main(InsuranceWebServicesClient.java:748)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:85)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:58)
 at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:60)
 at java.lang.reflect.Method.invoke(Method.java:391)
 at com.ibm.websphere.client.applicationclient.launchClient.
createContainerAndLaunchApp(launchClient.java:649)
```

If using the HTTP protocol, the following error is displayed:

```
Client caught exception getting the InsruanceWebServicesPort
using the URL
"http://svtlnx1:9081/WebSvcsInsSession20EJB/services/WSMultiProtocol"
java.net.MalformedURLException: no protocol:
"http://svtlnx1:9081/WebSvcsInsSession20EJB/services/WSMultiProtocol"
```

If using the JMS protocol, the following error is displayed:

```
Client caught exception getting the InsruanceWebServicesPort
using the URL
"jms:/queue?destination=jms/MultiProtocol_Q&connectionFactory=jms/InsuranceServices_Q

CF&targetService=WSMultiProtocolJMS&jndiProviderURL=IIOP://svtlnx1.austin.ibm.com:981
1"
java.net.MalformedURLException: no protocol:
"jms:/queue?destination=jms/MultiProtocol_Q&connectionFactory=jms/InsuranceServices_Q

CF&targetService=WSMultiProtocolJMS&jndiProviderURL=IIOP://svtlnx1.austin.ibm.com:981
1"
       at java.net.URL.<init> (URL.java(Compiled Code))
Making calls to methods in WSMultiprotocolWebServicesBean ...
```

Solution: To resolve the problem, update the jnlp.jsp file to remove the quotations (″ ″) from the argument. When a jnlp.jsp file has statements or expressions that begin with ″${″ and the statement is not to be interpreted as an expression, then add a backward slash ″\″, as shown in the following example:

```
<argument>-CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/DummyClientKeyFile.jks</argument>
<argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/DummyClientTrustFile.jks</argument>
```

For the EJB protocol, use the following example argument to correct the errors:

```
<argument>-url=wsejb:/com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome?jndiName=com/ibm/wssvt/tc
/pli/ejb/WSMultiProtocolHome&</argument>
```

For the HTTP protocol, use the following argument to correct the errors:

```
<argument>-url=http://svtaix23:9081/WebSvcsInsSession20EJB/services/WSMultiProtocol</argument>
```

For the JMS protocol, use the following argument to correct the errors:

```
<argument>-url=jms:/queue?destination=jms/MultiProtocol_Q&connectionFactory=
jms/InsuranceServices_QCF&targetService=
WSMultiProtocolJMS&jndiProviderURL=IIOP://svtaix23.austin.ibm.com:9811</argument>
```

Now, rerun the client from Java Web Start.

Client application Java Network Launcher Protocol deployment descriptor file

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application.

Location

The client application has an Application Clients run-time dependency that provides the following:

- Java 2 Runtime Environment from IBM
- Application Clients run-time properties
- SSL KeyStore and TrustStore file
- Application Clients run-time library JAR files (optional for Thin Application client applications)

If the Application Clients run-time dependency is not met, it is downloaded and installed in Java Web Start (JWS), as described by the Application Clients run-time installer JNLP descriptor file.

```
<j2se version="WASclient6.1.0"
href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJre/AppClientRT.jsp"/>
```

Usage notes

The client application must also include the `WebSphereClientLauncher.jar` file, which contains the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, that completes one of the following actions:

- If it is a J2EE Application client application (that is the resources for the application contain an EAR file with a client application), then the launcher class starts a second Java Virtual Machine (JVM) using the JRE provided by the Application Clients run-time dependency and launches the J2EE Application client application that is packaged in the EAR file.

  The EAR file must be specified as a JAR resource so that it can be downloaded to JWS and specified in the system property, `com.ibm.websphere.client.launcher.ear`. See “JNLP descriptor file for a J2EE Application client application” on page 251 for an example.
- If it is a Thin Application client application, then the launcher class uses the current JVM from the Application Clients run-time dependency and invokes the Thin Application client application main method.

  The Thin Application client application JAR file must be specified as a JAR resource so that it can be downloaded to JWS and the name of the class containing main method entry point is specified in the system property, `com.ibm.websphere.launcher.main`. See “JNLP descriptor file for a Thin Application client application” on page 253 for an example.

  Unlike the J2EE Application client application, the Thin Application client application is not loading the Application Clients run-time library JAR files from the Application Clients run-time dependency. It is downloaded from the server directly as it is for the Thin Application client application JAR file. An Application Clients run-time library component JNLP descriptor is used for specifying the Application Clients run-time library JAR files resources, as shown in the following example:

  ```
  <extension name="WAS Thin EJB Client Library"
                    href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/AppClientLib.jnlp"/>
  ```

  The JNLP specification requires all the resource (JAR or EAR) files used in a JNLP file to be signed.

  You can specify the –CC arguments defined in the launchClient tool for a J2EE Application client application in application arguments section of the JNLP descriptor files. However, only –CCD is supported for a Thin Application client application to define system properties and the JNLP <property> tag can also be used to define system properties. See the following example for details:

  ```
  <property name="java.naming.provider.url" value="corbaloc:iiop:myserver.com:9089"/>
  ```

  For a J2EE Application client application, specify the following application arguments as defined in the JNLP.

  1. Specify your target server provider URL, as shown in the following example:

     ```
     <argument> >-CCDjava.naming.provider.url =corbaloc:iiop:myserver.mydomain.com:9080 </argument>
     ```
  2. Specify the SSL Key File and SSL Trust File location. These files are expected to be available in the client machine. To use the ones in the Application Clients run-time dependency installed in JWS cache, specify these application arguments:

```
<argument> -CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12 </argument>
<argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12 </argument>
```

3. Specify the initial naming context factor, as shown in the following example:

   ```
   <argument>-CCDjava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory </argument>
   ```

   For a Thin Application client application, you also need to specify the actual location of the `sas.client.props` and `ssl.client.props` files located in the Application Clients run-time dependency that is installed in the JWS cache.

   ```
   <argument>-CCDcom.ibm.CORBA.ConfigURL=file:${WAS_ROOT}/properties/sas.client.props </argument>
   <argument>-CCDcom.ibm.SSL.ConfigURL=file:${WAS_ROOT}/properties/ssl.client.props </argument>
   ```

   If any of the default settings in the `sas.client.props` and `ssl.client.props` file need modifying, use the –CCD to change the settings through the system properties, as shown in the following example:

   ```
   <argument>-CCDjavacom.ibm.CORBA.securityEnabled=false </argument>
   ```

   Note: The ${WAS_ROOT} token used in the JNLP file is replaced by the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, to the actual location of the Application Clients run-time dependency installation in the JWS cache. If you are using JSP to dynamically create this JNLP description file, you must escape this token because it has a different meaning in JSP 2.0. See the following example for details:

   ```
   <argument>-CCDcom.ibm.ssl.keyStore=\${WAS_ROOT}/etc/key.p12 </argument>
   <argument>-CCDcom.ibm.ssl.trustStore=\${WAS_ROOT}/etc/trust.p12 </argument>
   ```

JNLP descriptor file for a J2EE Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application. If it is a J2EE Application client application (that is the resources for the application contain an EAR file with a client application), then the launcher class starts a second Java Virtual Machine (JVM) using the JRE provided by the Application Clients run-time dependency and launches the J2EE Application client application that is packaged in the EAR file.

Here is an example of the client application JNLP descriptor file for a J2EE Application client application.

```
<!--
"This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for
redistribution by customer, as part of such an application, in customer's
own products."

Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2004
All Rights Reserved * Licensed Materials - Property of IBM
–->
<!––
 This is a generic jnlp for a client app.  It will specify the WAS JRE
  as a dependency as well as the client launcher
––>
<!-- ========================================================================= -->
<!-- TODO: change "codebase" to the actual url location of this jnlp file -->
<!-- ========================================================================= -->
<jnlp spec="1.0+"
 codebase="http://your_server:port_number/J2EEWebStartWeb/JavaClients/WebStartExample">

  <information>
    <title>J2EE Client Exampletitle>J2EE Client Example</title>
    <vendor>IBM Client Teamvendor>IBM Client Team</vendor>
    <description>JVE WebStart example of: J2EE Client Example</description>
    <description kind="short">J2EE Client Example</description>
    <description kind="tooltip">J2EE Client Example</description>
  </information>

  <security>
```

```
    <all-permissions/>
  </security>

  <resources>

    <!-- ========================================================================= -->
    <!-- TODO: Update the "version" value to match your Application Client runtime -->
    <!--       Update the "href" value to the url for downloading the Application  -->
    <!--       Client runtime.                                                     -->
    <!-- ========================================================================= -->
    <j2se version="WASClient6.1.0"
     href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJre/AppClientRT.jsp"/

    <!-- The client app will require a client launcher -->
    <jar href="../Launcher/WebSphereClientLauncher.jar" main="true"/>

    <!-- Ear we want to download to the client -->
    <jar href="J2eeJWS.ear"/>

    <!-- The launcher depends on this property to be set -->
    <property name="com.ibm.websphere.client.launcher.ear" value="J2eeJWS.ear"/>

  </resources>

  <!-- WebStart will consider the Launcher as the app. to run  -->
  <application-desc main-class="com.ibm.websphere.client.launcher.ClientLauncher">
    <argument>-CCproviderURL=corbaloc:iiop:your_server</argument>
    <argument>-CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12</argument>
    <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12</argument>
  </application-desc>
</jnlp>
```

JNLP descriptor file for a Thin Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application. If it is a Thin Application client application, then the launcher class uses the current JVM from the Application Clients run-time dependency and invokes the Thin Application client application main method.

Here is an example of the JNLP descriptor file for a Thin Application client application.

```
<!--
"This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for
redistribution by customer, as part of such an application, in customer's
own products."

Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2004
All Rights Reserved * Licensed Materials - Property of IBM
-->

<!--
   This is a generic jnlp for a client app.  It will specify the WAS JRE
   as a dependency as well as the client launcher
 -->

<!-- ========================================================================= -->
<!-- TODO: change "codebase" to the actual url location of this jnlp file -->
<!-- ========================================================================= -->
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://your_server:port_number/J2EEWebStartWeb/JavaClients/WebStartExample">
  <information>
    <title>Thin Basic Calculator Client Samples</title>
```

```
    <vendor>IBM</vendor>
    <description>Thin Basic Calculator Client Samples</description>
    <offline-allowed/>
  </information>

  <security>
    <all-permissions/>
  <security>

  <resources>
    <j2se version="WASClient6.1.0"
     href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJre/AppClientRT.jspz"/>


    <extension name="WAS Thin EJB Client Library"
               href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/AppClientLib.jnlp"/>

   <!-- you must use the jar resource for JWS LaunchClient class here if using JWS LaunchClient
 wrapper launcher -->
    <jar href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/WebSphereClientLauncher.jar" main="true"/>

    <jar href="BasicCalculatorClientCommon.jar"/>
    <jar href="BasicCalculatorEJB.jar"/>
    <jar href="BasicCalculatorThinClient.jar"/>

    <property name="com.ibm.websphere.client.launcher.main"
value="com.ibm.websphere.samples.technologysamples.basiccalcthinclient.BasicCalculatorClientThinMain"/>
  <property name="java.naming.factory.initial"
            value="com.ibm.websphere.naming.WsnInitialContextFactory" />
  <property name="java.naming.provider.url"
            value="corbaloc:iiop:your_server:port_number"/>
  <property name="com.ibm.CORBA.ConfigURL"

value="http://your_server:port_number/J2EEWebStartWeb/JavaClients/sas.client.props"/>
  <property name="com.ibm.SSL.ConfigURL"

value="http://your_server:port_number/J2EEWebStartWeb/JavaClients/ssl.client.props"/>

  <!-- *** Logging Properties ***
       <property name="com.ibm.websphere.client.launcher.jws.trace" />
       <property name="java.util.logging.configureByServer" value="true" />
       <property name="traceSettingsFile" value="TraceSettings.properties" />
       <property name="com.ibm.CORBA.Debug" value="true" />
       <property name="com.ibm.CORBA.CommTrace" value="true" />
       <property name="java.util.logging.manager" value="com.ibm.ws.bootstrap.WsLogManager" />
       <property name="com.ibm.CORBA.RasManager" value="com.ibm.websphere.ras.WsOrbRasManager" />
  -->
  </resources>
    <application-desc>
      <argument>-CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12</argument>
      <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12</argument>
      <argument>add</argument>
      <argument>1</argument>
      <argument>2</argument>
  </application-desc>
</jnlp>
```

ClientLauncher class:

The class, com.ibm.websphere.client.installer.ClientLauncher, contains a main() method that is called by Java Web Start (JWS) to launch the client application. The Java Web Start client is used with platforms that support a Web browser.

This client is packaged in the `WebSphereClientLauncher.jar` file that is located in a WebSphere Application Server clients installation under the *<app_server_root>*/ JWS directory.

This launcher class configures the run-time environment for J2EE application clients and thin client applications (not J2EE application clients).

The launcher class requires that the following properties are defined. These properties are not defined in a separate properties file. Instead, the properties are defined as part of the Java Network Launching Protocol (JNLP) files.

com.ibm.websphere.client.launcher.main
If the client application is a Thin Application client, then this property should be specified. It specifies the class where the main entry point of the client application resides.

com.ibm.websphere.client.launcher.ear
If the client application is a J2EE Application client, then this property should be specified. It specifies the name of the EAR file to be executed. This property takes precedence over com.ibm.websphere.client.launcher.main. However, only one of the two properties should be specified.

Launcher tool:

The launcher configures the run-time environment for J2EE application clients and thin client applications (not J2EE application clients). The launcher utility is located in the main entry point of the Java Network Launching Protocol (JNLP) application client. The main class launcher name is com.ibm.websphere.client.launcher.ClientLauncher and is located in the `WebSphereClientLauncher.jar` file.

The launcher tool requires that the following properties are defined.

com.ibm.websphere.client.launcher.main
If the client that is to be run is a thin client, then this property should be specified. It specifies the class where the main entry point of the application resides.

com.ibm.websphere.client.launcher.ear
If the client that is to run is the J2EE client, then this property should be specified. It specifies the name of the ear file to be executed. This property takes precedence over com.ibm.websphere.client.launcher.main although only one of the two properties should be specified.

com.ibm.websphere.client.launcher.classpath.* (required for J2EE client applications only)
There can be a set of properties that are prefixed with com.ibm.websphere.client.launcher.classpath. Each property specifies a JAR file that is to be added to the class path of the application. This JAR file is a JAR file that is already defined as a resource for the application. This file is needed so that the correct elements of the class path of the Java Virtual Machine (JVM) starting the client launcher can be retrieved and added to the class path of the (JVM) that is to be spawned for the application client.

These properties are not defined in a separate properties file. Instead, they are defined as part of the Java Network Launching Protocol files.

Preparing the Application Client run-time dependency component for Java Web Start

For a J2EE application client application and or Thin application client application to be launched using Java Web Start (JWS), an Java Runtime Environment that IBM provides, the library JAR files and properties files bundled in Application Client for WebSphere Application Server must be installed in the JWS. This article provides the steps to build the Application Client run-time dependency component from an Application Client installation. It is packaged as a Web Archive Resource (WAR) file that can be installed in an Application Server.

Install the Application Client for WebSphere Application Server for the platform to which the client application deploys. If there is a requirement to deploy the client application to multiple platforms, the Application Client run-time dependency component must be built separately for each platform that client application supports.

For example, if the client application deploys to both the Windows platform and Linux platform, follows the steps for this task to build the Application Client run-time dependency component for Windows on a Windows platform machine with the Application Client for WebSphere Application Server for Windows installed. Now, repeat the steps for this task to build the Application Client run-time dependency component for Linux on a Linux platform machine with the Application Client for WebSphere Application Server for Linux installed.

1. Install the Application Client for WebSphere Application Server for the client application supported operating systems. Install Application Client in the `C:\Program Files\IBM\WebSphere\AppClient` directory.
2. Change the directory to the installation bin directory. See the following example for help:

   ```
   CD C:\Program files\IBM\WebSphere\AppClient\bin
   ```
3. Run the buildClientRuntime tool to generate the Application Client run-time JAR file in a temporary directory which contains the Java 2 Runtime Environment, Application Client run-time properties, the SSL KeyStore and TrustStore file, and the Application Client run-time library JAR files. See the following example for help:

   ```
   buildClientRuntime C:\WebApp1\runtime\WASClient6.1_windows.jar
   ```

   If you are building an Application Client run-time JAR file only for serving Thin application client applications and not for J2EE application client applications, you can reduce the size of the generated JAR file by not including the Application Client run-time library JAR files. An extra parameter is passed to the buildClientRuntime tool, as the following example shows:

   ```
   buildClientRuntime C:\WebApp1\runtime\WASClient6.1_windows.jar
                      buildThin
   ```
4. Copy the `WebSphereClientRuntimeInstaller.jar` file to the same location of the JAR file generated in the previous step. This JAR file is located in the JWS directory of the WebSphere Application Server clients installation. See the following example for help:

   ```
   copy ..\JWS\WebSphereClientRuntimeInstaller.jar   C:\WebApp1\runtime
   ```
5. Sign the JAR files created from the previous steps, using the Java 2 SDK jarsigner utility, as the following example shows:

   ```
   cd C:\WebApp1\runtime

   jarsigner -keystore myKeystore -storepass myPassword
                   WASClient6.1_windows.jar myKeyAliasName

   jarsigner -keystore myKeystore -storepass myPassword
                   WebSphereClientRuntimeInstaller.jar myKeyAliasName
   ```

   a. This step also requires you to create a keystore file, such as `myKeystore`.

   b. You must also create a self-signed certificate for the `myKeystore` file.

   Note: When running the JAR signer tool on HP platforms, add the `-J"-Xmx256m"` flag to the jarsigner command to increase the available heap size and prevent the error, `OutOfMemoryError`. See the following example for help:

   ```
   jarsigner -J"-Xmx256m" -keystore myKeystore -storepass myPassword
                      WebSphereClientRuntimeInstaller.jar myKeyAliasName
   ```
6. Create an Application Client run-time installer JNLP descriptor file or a JavaServer Pages (JSP) file if it is generated dynamically in the same temporary directory as previous step. See the sample JNLP file shown in the Example section of this topic.
7. Package the two signed JAR files and the Application Client run-time installer JNLP descriptor file into a WAR file. This WAR file is packaged into an EAR file that can be deployed to an Application Server.

Your Web application is ready to serve the Application Client run time and the JRE environment.

```
<!-- This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer (a) for its own instruction and study, (b) in order
to develop applications designed to run with an IBM WebSphere product, either for customer's
own internal use or for redistribution by customer, as part of such an application, in
customer's own products.

Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2005
All Rights Reserved * Licensed Materials - Property of IBM
-->

<%-- // to set the Last_Modified header so that the JNLP client will know whether to download
     // the JNLP file again and update the cached copy.
     String jspPath = application.getRealPath(request.getServletPath());
     java.io.File jspFile = new java.io.File(jspPath);
     long lastModified = jspFile.lastModified();
%><%
      // locally declared variables
      String url=request.getRequestURL().toString();
      String jnlpCodeBase=url.substring(0,url.lastIndexOf('/'));
      String jnlpRefURL=url.substring(url.lastIndexOf('/')+1,url.length());

      // Need to set a JNLP mime type - if WebStart is installed on the client,
      // this header will induce the browser to drive the WebStart Client
      response.setContentType("application/x-java-jnlp-file");                          1
      response.setHeader("Cache-Control", null);
      response.setHeader("Set-Cookie", null);
      response.setHeader("Vary", null);
      response.setDateHeader("Last-Modified", lastModified);

      // An installer must reply with the version number for a given install
      if (response.containsHeader("x-java-jnlp-version-id"))
        response.setHeader("x-java-jnlp-version-id", "WASClient6.1.0");           2
      else
        response.addHeader("x-java-jnlp-version-id", "WASClient6.1.0");
%>

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

<!-- ========================================================== -->
<!-- TODO: change "codebase" to the actual url location of this jsp -->
<!-- ========================================================== -->

<jnlp spec="1.0+"
codebase="http://YOUR_APP_SERVER:PORTNUMBER/WEBAPP_CONTEXT_ROOT/Runtime/WebSphereJre">

<information>
  <title>Application Client Java Runtime Environment</title>
  <vendor>IBM</vendor>
  <icon href="icon.gif"/>
  <description>Application Client Java Runtime Environment</description>
  <description kind="short">Applicaiton Client JRE</description>
  <description kind="tooltip">Applicaiton Client JRE</description>
  <offline-allowed/>
</information>

<security>
 <all-permissions/>
</security>

 <resources>
  <j2se version="1.4+"/><%-- The installer can use any 1.4 JRE --%> 3
   <jar href="WebSphereClientRuntimeInstaller.jar" main="true"/> 4

  <!-- JRE version registration with Web Start -->
  <property name="com.ibm.websphere.client.jre.version" value="WASClient6.1.0"/> 5
 </resources>
```

```
 <resources os="Windows"> 6
<!-- ============================================================ -->
<!-- TODO: the property value for unix platform is "java/jre/bin/javaw" -->
<!--   and the "os" value match to your target client machine platform             -->
<!-- ============================================================ -->

   <jar href="WASClient6.1.0_Windows.jar"/> 7

<!-- ============================================================ -->
<!-- TODO:  property value for unix platform is "java/jre/bin/javaw" -->
<!-- ============================================================ -->
<!-- relative path of the jre executable —->

   <property name="com.ibm.websphere.client.jre.launch.java"
 value="java\jre\bin\javaw.exe"/> 8

 </resources>
 <installer-desc main-class="com.ibm.websphere.client.installer.ClientRuntimeInstaller"/>
</jnlp>
```

1. Specifies that the file is a JNLP mime type so that the browser can process the JNLP file.
2. Specifies the exact version of this Application Client run-time dependency component in the response by setting the HTTP header field: `x-java-jnlp-version-id`.
3. Specifies the required JRE version to run the installer program.
4. Specifies the installer `WebSphereClientRuntimeInstaller.jar` file, which contains the ClientRuntimeInstaller class.
5. Specifies a system property that defines the version of Application Client run-time dependency component. This version is registered to the JNLP client.
6. Specifies resources for a particular platform. Each supported client application platform needs its own separate JAR file.
7. Specifies the Application Client run-time dependency component JAR file.
8. Specifies the program to call that starts a JVM for the client application.

Preparing Application Client run-time library component for Java Web Start.

buildClientRuntime tool:

For a J2EE application client application and or Thin application client application to be launched using Java Web Start (JWS), the library JAR files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to build those JAR files. The Java Web Start client is used with platforms that support a Web browser. .

The buildClientRuntime tool builds the required components from the WebSphere Application Server clients installation into the JAR file specified on the command. This JAR file contains:

- License files
- Java 2 Runtime Environment (JRE) that IBM provides
- Application Clients run-time properties and configuration
- SSL KeyStore and TrustStore files
- Run-time library JAR files

In the case of building an Application Clients run-time JAR file only for serving Thin Application client applications and not for J2EE Application client applications, the run-time library JAR files and the Application Clients run-time properties files are not included, except the configuration files, `sas.client.props`, `ssl.client.props` and `soap.client.props`, located in the `WAS_ROOT/properties` directory. The Java Web Start client is used with platforms that support a Web browser.

The command-line invocation syntax for the buildClientRuntime tool is shown in the following example:

```
Windows Usage:    buildClientRuntime .bat  jar_file  [type]
Unix Usage:         buildClientRuntime.sh    jar_file  [type]
Where:
        jar_file
Specifies the target jar file name.

Range:
                    buildJ2EE - Default value that builds a Application Clients
                                run-time library for J2EE application.
                    buildThin - Builds a Application Clients run-time library
                                for Thin application.
```

ClientRuntimeInstaller class: This class, com.ibm.websphere.client.installer.ClientRuntimeInstaller, contains a main() method that Java Web Start (JWS) calls to install the Application Client for WebSphere Application Server run-time dependency component in JWS cache. It is packaged in `WebSphereClientRuntimeInstaller.jar` file located in the Application Client for WebSphere Application Server installation in the *<app_server_root>*`/JWS` directory.

Specify the `WebSphereClientRuntimeInstaller.jar` file and the Application Client run-time dependency component JAR file as JAR resources in the Application Client run-time installer Java Network Launcher Protocol (JNLP) descriptor file. See the following example for details:

```
<jar href="Launcher/WebSphereClientRuntimeInstall.jar" main="true"/>
<jar href="Launcher/WASClient6.1_windows.jarRuntimeInstall.jar" main="true"/>
```

The ClientRuntimeInstaller class main method requires the following properties to be set in the JNLP file:

com.ibm.websphere.client.jre.version
: Specifies a Java Runtime Environment (JRE) version name that is to be used when referring to the Application Client run-time dependency component.

com.ibm.websphere.client.jre.launch.java
: Specifies the relative location of the javaw.exe program in the Application Client run-time dependency component JAR file.

The previously mentioned properties, JRE version name and the location of the javaw.exe program are registered to the Java Web Start Application Manager, as shown in the following example:

```
<property name="com.ibm.websphere.client.jre.version" value="java\jre\bin\javaw.exe"/>
<property name="com.ibm.websphere.client.jre.launch.java" value="WASclient6.1"/>
```

Preparing Application Clients run-time library component for Java Web Start

The Java Web Start client is used with platforms that support a Web browser. For a Thin Application client application to be launched using Java Web Start (JWS), you also need to create a Java Network Launching Protocol (JNLP) component to serve the Application Clients run-time library JAR files from the Application server. This JNLP component is referenced in the client application JNLP file with the <extension> tag. This article provides the steps to build the Application Clients run-time library component from an Application Clients installation. It is packaged as its own Web Archive Resource (WAR) file or to the same WAR file that contains the Application Clients run-time dependency component, and can be installed in an Application server.

Install the Application Client for WebSphere Application Server for the platform to which client applications deploy.

1. Install the Application Clients on the client application supported operating system. For example, install Application Clients in the `C:\Program Files\IBM\WebSphere\AppClient` directory.
2. Change the directory to the installation bin directory. For example:

   ```
   CD C:\Program files\IBM\WebSphere\AppClient\bin
   ```

3. Run `buildClientLibJars` to copy the Application Clients run-time library JAR files from the Application Clients installation to a temporary directory. All the JAR files in the temporary directory are signed, as shown in the following example:

   ```
   buildClientLibJars C:\WebApp1\runtime\WebSphereJars
                      myKeystore myPassword myKeyAliasName
   ```

   a. This step also requires you to create a keystroke file, such as `myKeystore`.

   b. You must also create a self-signed certificate for the `myKeystore` file.

4. Create an Application Clients run-time installer JNLP descriptor file in the same temporary directory as the previous step. See the sample JNLP file shown in the Example section of this topic.

5. Package these JAR files and the Application Clients run-time library component JNLP descriptor file into a WAR file. You can also package both Application Clients run-time library component and Application Clients run-time dependency component in the same WAR file. This WAR file is packaged into an EAR file that can deployed to an Application server.

```
<!--
  "This sample program is provided AS IS and can be used, executed, copied
and modified without royalty payment by customer (a) for its own instruction
and study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for redistribution
by customer, as part of such an application, in customer's own products."
Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2005
All Rights Reserved * Licensed Materials - Property of IBM
-->
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://YOUR_APP_SERVER:PORTNUMBER/WEBAPP_CONTEXT_ROOT/Runtime/WebSphereJars">
 <information>
  <title>Application Client Library</title>
  <vendor>IBM</vendor>
  <icon href="icon.gif"/>
  <description>Application Client Library</description>
  <description kind="short">Application Client Library</description>
  <description kind="tooltip">Application Client Library</description>
  <offline-allowed/>
 </information>

<security>
 <all-permissions/>
</security>

<component-desc/>
 <resources><jar href="activation-impl.jar"/>
<jar href="bootstrap.jar"/>
<jar href="com.ibm.events.client_6.1.0.jar"/>
<jar href="com.ibm.mq.jar"/>
<jar href="com.ibm.mqjms.jar"/>
<jar href="com.ibm.uddi.client_1.0.0.jar"/>
<jar href="com.ibm.ws.bootstrap_6.1.0.jar"/>
<jar href="com.ibm.ws.debug.osgi_6.1.0.jar"/>
<jar href="com.ibm.ws.emf_2.0.0.jar"/>
<jar href="com.ibm.ws.j2ee.client_6.1.0.jar"/>
<jar href="com.ibm.ws.runtime.dist_6.1.0.jar"/>
<jar href="com.ibm.ws.runtime.gateway_6.1.0.jar"/>
<jar href="com.ibm.ws.runtime_6.1.0.jar"/>
<jar href="com.ibm.ws.security.crypto_6.1.0.jar"/>
<jar href="com.ibm.ws.sib.client_2.0.0.jar"/>
<jar href="com.ibm.ws.sib.utils_2.0.0.jar"/>
<jar href="com.ibm.ws.wccm_6.1.0.jar"/>
<jar href="com.ibm.wsspi.extension_6.1.0.jar"/>
<jar href="dhbcore.jar"/>
<jar href="j2ee.jar"/>
<jar href="launchclient.jar"/>
<jar href="lmproxy.jar"/>
<jar href="mail-impl.jar"/>
```

```
<jar href="org.eclipse.core.runtime_3.1.1.jar"/>
<jar href="org.eclipse.osgi_3.1.1.jar"/>
<jar href="org.eclipse.update.configurator_3.1.0.jar"/>
<jar href="properties.jar"/>
<jar href="serviceadapter.jar"/>
<jar href="startup.jar"/>
<jar href="urlprotocols.jar"/>
<jar href="WebSphereClientLauncher.jar"/>
 <resources/><jnlp/>
```

buildClientLibJars tool:

For a J2EE application client application and or Thin application client application to be launched using Java Web Start (JWS), the properties files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to create those property JAR files. The Java Web Start client is used with platforms that support a Web browser.

The buildClientLibJars tool copies the JAR files from the Application Client for WebSphere Application Server installation and creates a `properties.jar` file, which contains the properties files from the Application Clients installation properties directory to a specified location. When this property is created, the tool uses the value of `keystore`, `storepass` and `alias` to sign all the JAR files in the specified location.

```
Windows Usage: buildClientLibJars.bat  target_dir keystore  storepass  alias
Unix Usage:        buildClientLibJars.sh  target_dir keystore  storepass  alias
Where:
     target_dir       Specifies the target directory where the Application
                           Clients library JAR files copied to.
    keystore         Specifies a keystore file.
    storepass       Specifies the keystore password.
    alias               Specifies an alias for the key object in the key file.
```

Using the Java Web Start sample

The EAR file, `WebSphereClientRuntime.ear`, is provided in the JWS directory of the Client Application for WebSphere Application Server installation. This EAR file provides a sample Application Clients run-time installer JNLP descriptor file and a sample Application Clients run-time library component JNLP descriptor file. Follow the steps in this task to build the Application Clients run-time dependency component and the Application Clients run-time library component. Add these components to the `WebSphereClientRuntime.ear` file, and then install the EAR file in an Application Server to be used by the client application.

Install the Application Client for WebSphere Application Server for the platform to which the client application deploys. If there is a requirement to deploy the client application to multiple platforms, the Application Clients run-time dependency component must be built separately for each platform that the client application supports.

1. Install the Application Clients on the client application supported operating system. For example, install Application Clients in the `C:\Program Files\IBM\WebSphere\AppClient` directory.
2. Create the following temporary working directories:

   ```
   MKDIR C:\WebApp1
   MKDIR C:\WebApp1\runtime
   MKDIR C:\WebApp1\runtime\Widnows
   MKDIR C:\WebApp1\runtime\WebSphereJars
   ```

3. Change directory to the installation bin directory. See the following example for help:

   ```
   CD C:\Program files\IBM\WebSphere\AppClient\bin
   ```

4. Run the buildClientRuntime tool to generate the Application Clients run-time JAR file in a temporary directory that contains the Java 2 Runtime Environment that IBM provides, Application Clients run-time properties, the SSL KeyStore and TrustStore files, and the Application Clients run-time library JAR files. See the following example for details:

   ```
   buildClientRuntime C:\WebApp1\runtime\windows\WASClient6.1.0_Windows.jar
   ```

5. Copy the WebSphereClientRuntimeInstaller.jar file to the same location of the JAR file generated in the previous step. This JAR file is located in the JWS directory of the Application Client for WebSphere Application Server installation. For example, copy the ..\JWS\WebSphereClientRuntimeInstaller.jar file to the C:\WebApp1\runtime directory.
6. Sign the JAR files created from the previous steps, using the Java 2 SDK jarsigner utility. See the following example for details:

```
cd C:\WebApp1\runtime

jarsigner -keystore myKeystore -storepass myPassword
                WASClient6.1_windows.jar myKeyAliasName

jarsigner -keystore myKeystore -storepass myPassword
                WebSphereClientRuntimeInstaller.jar myKeyAliasName
```

   a. This step also requires you to create a keystore file, such as myKeystore.
   b. You must also create a self-signed certificate for the myKeystore file.
7. Run buildClientLibJars to copy the Application Clients run-time library JAR files from the Application Client for WebSphere Application Server installation to a temporary directory. All the JAR files in the temporary directory are signed. See the following example for details:

```
buildClientLibJars C:\WebApp1\runtime\WebSphereJars
                   myKeystore myPassword myKeyAliasName
```

   a. This step also requires you to create a keystroke file, such as myKeystore.
   b. You must also create a self-signed certificate for the myKeystore file.
8. Add all the JAR files created in the previous steps in the C:\WebApp1 directory to the WAR file within the WebSphereClientRuntime.ear file. The contents of the WAR file are shown in the following example:

```
The root of the WAR
├──META-INF
│                MANIFEST.MF
│
├──Runtime
│   ├──WebSpherejars
│   │          AppClientLib.jnlp
│   │          com.ibm.ws.runtime_6.1.0.jar
│   │          com.ibm.ws.j2ee.client_6.1.0.jar
│   │          com.ibm.ws.wccm_6.1.0.jar
│   │                    :
│   │          (all the jars created in step 7 under
│   │          c:\WebApp1\Runtime\WebSphereJars)
│   └──WebSphereJre
│             AppClientRT.jsp
│             WASClient6.1.0_Windows.jar
│             WebSphereClientLauncher.jar
│             WebSphereClientRuntimeInstaller.jar │
│
└──WEB-INF
        ibm-web-bnd.xmi
        ibm-web-ext.xmi
        web.xml
```

9. Install the WebSphereClientRuntime.ear file to an Application Server. You have just created an Application Clients run-time dependency component and Application Clients run-time libraries for serving J2EE Application client applications and Thin Application client applications using Java Network Launching Protocol (JNLP) or Java Web Start (JWS).

Installing Java Web Start

This topic provides the steps necessary to install JWS on the AIX platform.

Before you begin this task, see the following topics to understand Java Web Start (JWS) technology and its components:

- Prepare the Application Clients run-time dependency component for JWS

- Prepare the Application Clients run-time library component for JWS

Note: You can use Java Web Start on Java 2 Standard Edition Developer Kits that IBM provides, packaged in the Application Client for WebSphere Application Server, Version 6; Java Web Start on Sun Microsystems J2SE Software Development Kit or J2SE Java Runtime Environment 1.4.2, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems, or the Java Web Start on HP SDK or RTE for Java 2 version 1.4.2, which you can download from the HP Web site.

Use the following steps to install JWS on the AIX platform.

1. Install IBM Application Client for WebSphere Application Server.
2. Change your directory to the `client_install_root/java/jre` path.
3. Run `sh bin/webstart_install_sdk.sh`.
4. When prompted for the Java path, enter your JRE path. Use the following example: `client_install_root/java/jre`.
5. You should see the following messages, which indicate that JWS installs successfully:
   - Obtaining version...
   - You appear to be running 1.4.2
   - Extracting...
   - Updating ~/.mailcap...
   - Updating ~/.mime.types...
6. Change your path to the JWS installed path. For example, enter `client_install_root/java/jre/javaws/`.
7. Run `./javaws` from the path mentioned in the previous step.

Java Web Start for Application client best practices

Do not use JSP to dynamically generate a JNLP file, otherwise the JNLP jsp page cannot be opened in some IE browsers. To use a static JNLP file, you will need to add the following mime type mapping in the web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
       <display-name>
       WAS Client runtime for Java Web Start</display-name>
       <welcome-file-list>
              <welcome-file>index.html</welcome-file>
              <welcome-file>index.htm</welcome-file>
              <welcome-file>index.jsp</welcome-file>
              <welcome-file>default.html</welcome-file>
              <welcome-file>default.htm</welcome-file>
              <welcome-file>default.jsp</welcome-file>
       </welcome-file-list>
       <mime-mapping>
       <extension>jnlp</extension>
       <mime-type>application/x-java-jnlp-file</mime-type>
       </mime-mapping>
</web-app>
```

Installing Application Client for WebSphere Application Server

This topic describes how to install the Application Client for WebSphere Application Server using the installation image on the product CD-ROM.

Running client applications that communicate with a WebSphere Application Server requires that elements of the Application Server are installed on the system on which the client applications run. However, if the system does not have the Application Server installed, you can install Application Client, which provides a stand-alone client run-time environment for your client applications. See the Supported Prerequisites page for more information on supported product platforms.

The steps that follow provide enough detail to guide you through preparing for, choosing, and installing the variety of options and features provided. To prepare for installation and to make yourself familiar with installation options, complete the steps in this article and read the related topics, before you start to use the installation tools. Specifically, read these topics before installing the product:

- Installing silently
- Best practices for installing

As a general rule, if you launch an installation and there is a problem such as not having enough temporary space or not having the right packages on your Linux or UNIX systems, then cancel the installation, and make the required changes. Restart the installation to initiate your changes.

You can install this product by a non-root user on a UNIX operating system and non-administrator user on a Windows operating system. However, the following functions are not enabled on Windows if the product is installed by a non-administrator user:

- ActiveX to EJB bridge
- Applet Client
- Launching Java Web Start from browser

In Version 6.x, the Application Client for WebSphere Application Server is installable on a machine with a previous version of Application Clients. However, you cannot install a Version 6.x Application Client on top of a previous version of the Application Client. For example, if a Version 5 Application Clients install under the `C:\WebSphere\AppClient` directory, you can not choose the same install location for your V6.x Application Clients installation.

Note: For Application Clients to coexist, there is a limitation on Applet client and ActiveX client on Windows that can not be coexisted with V5.0.x and V4.x of the clients. For example, the Applet client feature in Version 6.x cannot coexist with the Applet client feature in any previous release. This coexistence is not available because the installation of Applet client feature in Version 6.x sets the browser default Java Virtual Machine (JVM) using the Java Runtime Environment (JRE) from the Version 6.x installation, which is Java Runtime Environment Version 1.4.2. Similarly, the ActiveX to EJB Bridge feature in Version 6.x sets the Windows system path to use the JRE from the Version 6.x installation.

1. Install Application Client for WebSphere Application Server using the launchpad.

   See Install Application Client for WebSphere Application Server using the launchpad.

   Linux The launchpad program is available on the root directory of the product CD in the program, `launchpad.sh`.

   Windows The launchpad program is available on the root directory of the product CD in the program `launchpad.bat`.

   Note: The free download Application Client installation is not packaged with the launchPad program.

   a. Click **Launch the installation wizard for Application Client for WebSphere Application Server** from the launchpad tool to launch the InstallShield for MultiPlatforms installation wizard. This action launches the installation wizard.

      The Readme documentation to which the launchpad links is the `readme.html` file in the CD root directory. The readme directory off the root of the CD has more detailed Readme files. The Installation Guide is in the `/docs` directory of the CD root directory.

Note: Readme file names are based on product offerings.

When you install application clients, the current working directory must be the directory where the installer binary program is located. This placement is important because the resolution of the class files location is done in the current working directory. For example:

```
cd /install_root/AppClient

./install
```

or

```
 cd <CD mount point>/AppClient

     ./install
```

Failing to use the correct working directory can cause ISMP errors that abort the installation.

The installation wizard does not upgrade or remove previous Application Clients installation automatically. Application Client V6.1 can be installed together with previously installed Application Clients if their versions are lower than 6.1. However, only one instance of Application Client V6.1 can be installed on the same system. When the installer is launched, it will detect any existing installation of Application Client V6.1 and direct the install flow to the feature panels so that additional features can be added on top of the existing installation.

b. As indicated in the previous example, you can start the installation wizard from the product CD-ROM, using the command line.

   On other Linux platforms and UNIX-based platforms, run the `./install` command.

   On Windows platforms, run the `Install.exe` command.

c. You can also perform a silent installation using the `-options responsefile -silent` parameter, which causes the installation wizard to read your responses from the options response file, instead of from the interactive graphical user interface. Customize the response file before installing silently. After customizing the file, issue the command to silently install. Silent installation is particularly useful if you install the product often.

   The rest of this procedure assumes that you are using the installation wizard. There are corresponding entries in the response file for every prompt that is described as part of the wizard. Review the description of the response file for more information. Comments in the file describe how to customize its options.

2. Click **Next** to continue when the Welcome panel is displayed.
   a. Click the radio button beside the **I accept the terms in the license agreement** message if you agree to the license agreement, and click **Next** to continue.
3. The installation wizard checks the system for prerequisites. Click **Next** if you see a success message on the wizard panel. If a warning message is displayed on the panel, click **Cancel** to exit the installation wizard and install the proper prerequisites to the system. **Note:** Application Client may be fully functional even if some prerequisites are not installed on the system, however it is recommended you update your system to the required level.
4. Specify a destination directory. Click **Next** to continue.
   a. Ensure that there is adequate space available in the target directory.
   b. Specify a target directory for the Application Client product.
   c. Enter the required target directory to proceed to the next panel. Deleting the default target location and leaving an installation directory field empty prevents you from continuing the installation process.
5. Choose a type of installation, and click **Next**.

   If you use the GUI you can choose a Typical installation type, which installs J2EE and Thin application client, Samples and IBM Developer Kit, Java 2 Technology Edition, or you can choose the custom installation type. For Windows, there are two custom installation types: Custom J2EE/Thin Client and Custom Pluggable Client. For other platforms, there is only one custom installation type.

Windows Custom J2EE/Thin Client

- If you select the **ActiveX to EJB Bridge** feature, then the following is displayed in a dialog box: Do you want to add Java runtime to the system path and make it the default JRE? If you answer Yes, then the Java run time is added to the beginning of the system path. If you answer No, then the ActiveX to EJB Bridge does not function from the Active Server Pages (ASP), unless you add the Java run time to the path. To add the Java run time later, see the topic ActiveX application clients or reinstall the Application Client.
- If you select the **Applet client** feature, then the following message might be displayed: An existing JDK or JRE has been detected on your computer. You chose to install the Applet Client, which will overwrite the registry entries for this JDK or JRE. Do you want to continue and install the Applet Client? If you select Yes, the installation overrides the registry on your machine. If you select No, the Applet client feature is not installed, and you are directed back feature dialog box.
- Install the Software Developer Kit feature, if you need to use any of the utilities that it provides, such as the javaccompiler, the jarutility or the jarsigner utility. The Java 2 Development Kit that IBM provides has two components, Java Runtime Environment (JRE) and a complete Software Developer Kit (SDK). The JRE sub feature is selected by default for the Custom J2EE/Thin Client installation type. The SDK component is optional; however, you must install the SDK component to compile the sample.
- Install the Administration Thin Client, if you need a runtime jar that is customed to enable client applications to perform WebSphere administration tasks. The Administration Thin Client is installed into *<install_root>/runtimes*.
- Install the Web Services Thin Client, if you need a runtime jar that is customed to enable client applications to communicate with the Application Server through Web Services. The Web Services Thin Client is installed into *<install_root>/runtimes*.

Windows Custom Pluggable Client

- Install the Pluggable Client samples if you want to make use of the Pluggable Client applications.

Linux Custom

- Install the Software Developer Kit feature, if you need to use any of the utilities that it provides, such as the *javaccompiler*, the *jarutility* or the *jarsigner* utility. The Java 2 Development Kit that IBM provides has two components, Java Runtime Environment (JRE) and a complete Software Developer Kit (SDK). The JRE sub feature is selected by default for the Custom J2EE/Thin Client installation type. The SDK component is optional; however, you must install the SDK component to compile the sample.
- Install the Administration Thin Client, if you need a runtime jar that is customized to enable client applications to perform WebSphere administration tasks. The Administration Thin Client is installed into *<install_root>/runtimes*.
- Install the Web Services Thin Client, if you need a runtime jar that is customed to enable client applications to communicate with the Application Server through Web Services. The Web Services Thin Client is installed into *<install_root>/runtimes*.

6. (Pluggable Client installation type only) Click **Next** to accept the detected Sun JRE, or click Browse to select the location of the installed Sun JRE. The Sun Software Development Kit installation location is optional. However, if the installation location is not provided, the installed Samples do not compile.
   - If Sun JRE has not been installed, the installation cannot be continued. Click **Cancel** to exit the installation. Install the Sun JRE, and restart the Pluggable Custom installation. The Sun JRE panel is displayed with the JRE path detected, and the Pluggable application client installation continues.
7. Enter the host name of the WebSphere Application Server machine. Click **Next** to continue. The default port number is 2809.
8. Review the summary information, and click **Next** to install the product code or you might also click **Back** to change your specifications.
9. Click **Finish** to exit the wizard, after the Application Client installs.

10. Verify the success of the installer program by examining the Completion panel and the <*install_root*>/logs/install/log.txt file for installation status. The installer program records the following indicators of success in the logs:
    - INSTCONFSUCCESS indicates that the installation is successful and that no further log analysis is required.
    - INSTCONFFAILED indicates an installation failure that you cannot retry or recover from without reinstalling.

You successfully installed the Application Client for WebSphere Application Server and the features you selected.

Use the installation verification utility to verify a successful installation. If the installation is not successful, fix the error as indicated in the installation error message. For example, if you do not have enough disk space, add more space, and reinstall the Application Client.

Best practices for installing Application Client for WebSphere Application Server

This topic provides tips for installing Application Client on multiple platforms.

The following table offers tips for installing Application Client on multiple platforms.

Operating environment	Tip
Linux and UNIX systems	Spaces are not supported in the name of the installation directory on Linux and UNIX platforms.
UNIX systems	When the application client installations are successful, the return code **0** is issued from the UNIX shell where you issued the /install command. Other return codes include: • **1** -- Failure • **2** -- Partial success Any other return code indicates an unsuccessful installation.
Solaris systems	Double-byte character set (DBCS) characters are not supported in the name of the installation directory on Solaris systems.
All platforms	Reserve at least 4 to 5MB free space in the target platform temporary directory.
All platforms	When specifying a different temporary directory while installing Application Client, the following message is displayed if the target platform default temporary directory does not have enough free space to install Application Client: `Error writing file =  There may not be enough temporary disk space. Try using -is:tempdir to use a temporary directory on a partition with more disk space.` Use the -is:tempdir installation option to specify a different temporary directory. For example, the following command uses /swap as a temporary directory during installation: `./install -is:tempdir /swap`

All platforms	After the installation, when changing the installation settings for the WebSphere Application Server host name and the port number, edit the `setupClient.bat` for Windows or `setupClient.sh` for UNIX. Change the `DEFAULTSERVERNAME` and `SERVERPORTNUMBER` to the new WebSphere Application Server host name and port number, respectively. If the `SERVERPORTNUMBER` is not set, then the default is 2809. Review the following example: `set DEFAULTSERVERNAME=NDServerName` `set SERVERPORTNUMBER=9810` The `setupClient.bat` file or `setupClient.sh` file is located in the `bin` sub-directory under the Application Client installation destination.

Installing Application Client for WebSphere Application Server silently

This topic provides the steps necessary to use the installation wizard and perform a silent installation.

Use these steps to perform a silent installation, which uses the installation wizard to install the product. Instead of displaying a user interface, the silent installation provides interaction between you and the wizard by reading all of your responses from a file that you must customize.

1. Verify that the user ID that you are using to run the silent installation has sufficient authority to perform the task.
2. Customize the option response file.
   a. Locate the sample options response file. The file name is `setup.response` in the operating system platform directory on the product CD-ROM.
   b. Make a copy to preserve the original response file. For example, copy the file as `myoptionsfile`.
   c. Edit the copy in your flat file editor of choice, on the target operating system. Read the directions within the response file to choose appropriate values.

      Note: To prepare the file for a silent installation on AIX, use UNIX line-end characters (0x0D0A) to terminate each line of the options response file.
   d. Make a non-commented option to have a silent install.
   e. Include custom option responses that reflect parameters for your system.
   f. Follow the instructions in the response file to choose appropriate values.
   g. Save the file.
3. Issue a command to use your custom response file: `Install.exe -options myoptionsfile -silent` for Windows platforms and `install -options ./myoptionsfile -silent` for Linux and UNIX platforms.

   The sample options response file is located in the `AppClient` directory on the product CD-ROM.
   a. Issue the following command from a command prompt to update your response file: `-OPT silentInstallLicenseAcceptance="true' .`

      Issuing this command indicates that you accept all IBM license terms associated with this product, which is necessary for installing application clients.
4. **Optional:** Restart your machine in response to the prompt that appears on Windows platforms when the installation is complete.

You installed application clients silently by using the response file.

To verify the silent install, look for the string `INSTCONFSUCCESS` in the log.txt file for successful installation and `INSTCONFFAILED` for a failed installation. For UNIX platforms, the install command returns a return code of **0** to indicate a successful installation, **1** to indicate failure and **2** to indicate partial success. Any other return code means that the installation failed.

When the InstallShield for MultiPlatforms (ISMP) fails and the log.txt file is not created, the error log file might have been created in one of the following directories:

- <*system_temp_dir*>/niflogs
- <*user_home*>/cltlogs

Uninstalling Application Client for WebSphere Application Server

This task describes using the uninstall program to uninstall the Application Client for WebSphere Application Server.

If you want to uninstall IBM Application Client for WebSphere Application Server manually, see the topic, Manually uninstalling on a Windows system.

1. Stop any browsers and any Java processes related to Application Client products.

   See Uninstalling the product.

2. Change directories to the `uninstall` directory before issuing the command to uninstall the application client. The command file is located in the *install_root*`/uninstall` directory on a Linux or z/OS platform, and in the *install_root*`\product\uninstall` directory on a Windows system.

   For example, to change directories before uninstalling the product from a Linux platform, issue this command if your installation root is `/opt/IBM/WebSphere/AppClient`:

   ```
   cd /opt/IBM/WebSphere/AppClient/uninstall
   ```

3. Issue the command to uninstall the product.

   Use the **uninstall** command.

   Linux The command file is named `uninstall`.

   Windows The command file is named `uninstall.exe`.

   Linux On Linux and z/OS platforms, issue the **uninstall** command from the *install_root*`/uninstall` directory:

   ```
   ./uninstall
   ```

   Windows On Windows platforms, call the **uninstall.exe** command:

   ```
   install_root\uninstall\uninstall.exe
   ```

   Windows Call the program directly from the *install_root*`\uninstall` directory. For example, if the installation root is `C:\IBM\WebSphere\AppClient`, issue the following command:

   ```
   C:\IBM\WebSphere\AppClient\uninstall> uninstall.exe
   ```

   The Uninstall wizard begins and displays the Welcome panel.

4. Click **Next** to begin uninstalling the product. The Uninstall wizard displays a Confirmation panel that lists the product and features that you are uninstalling.
5. Click **Next** to continue uninstalling the product. The Uninstall wizard deletes existing profiles first.

   After deleting profiles, the Uninstall wizard deletes core product files by component.

6. Click **Finish** to close the wizard after the wizard removes the product.

Application Client for WebSphere Application Server is uninstalled.

Verify the uninstall procedure by viewing the *install_root*`/logs/uninstall/log.txt` file for errors. Look for the `INSTCONFSUCCESS`, indicating a successful uninstall in the log file:

```
Uninstall, com.ibm.ws.install.ni.ismp.actions.ISMPLogSuccessMessageAction, msg1,
   INSTCONFSUCCESS
```

Deploying J2EE application clients on workstation platforms

You can deploy the J2EE application clients on workstation platforms using the methods described in this topic.

After developing an application client, deploy this application on client machines. *Deployment* consists of pulling together the various artifacts that the application client requires.

The *Application Client Resource Configuration Tool* (ACRCT) defines resources for the application client. These configurations are stored in the client `.jar` file within the application `.ear` file. The application client run time uses these configurations for resolving and creating an instance of the resources for the application client.

Note: This task only applies to J2EE application clients. Only perform this task if you configured your J2EE application client to use resource references.

1. Start the ACRCT and open an EAR file.
2. Configure new data source providers.
3. Configure mail providers and sessions.
4. Configure URL providers and sessions.
5. Configure Java messaging resources.
6. Configure new environment entries.
7. (Optional) Remove application client resources.
8. Save the EAR file.

Resource Adapters for the client

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

The resource adapter support for the J2EE client applications is a subset of the support for the server. For any resource adapter installed using the clientRAR tool, the client resource adapter is used in a non-managed environment and must conform to the J2EE Connector Architecture Specification Version 1.5 or higher. Only outbound connections to the EIS are supported through the ManagedConnectionFactory interfaces. The inbound messaging support (from the EIS), life cycle management, and work management aspects of the specification are not supported on the client.

For a client application to use a resource adapter, it must be installed in the directory specified by the environment variable, CLIENT_CONNECTOR_INSTALL_ROOT, defined when the setupCmdLine script runs. The launchClient tool, Application Client Resource Configuration Tool (ACRCT) and clientRAR tool all use this variable to find the default location of all installed resource adapters. To install a resource adapter in the client, use the clientRAR tool. Once the resource adapter is installed, it must be configured using the ACRCT. The client configuration tool adds the resource adapter configuration to the EAR file. Then, connection factories and administered objects are defined.

When running J2EE application clients, the `launchClient` script specifies a system property called `com.ibm.ws.client.installedConnector`, which is set to the same value as the `CLIENT_CONNECTOR_INSTALL_ROOT` variable. This is the default location for installed resource adapters and can be overridden for each `launchClient` call by specifying the `-CCD` parameter. When the client container is activated, all resource adapter subdirectories under the specified default location for the resource adapters directory are added to the classpath. This action allows the client application to use the resource adapters without using the ACRCT to specify any of the client resources.

Using resource adapters is a new mechanism for easily extending client applications.

Configuring resource adapters

Use the Application Client Resource Configuration Tool (ACRCT) to configure resource adapters.

1. Start the Application Client Resource Configuration Tool (ACRCT).

2. Open the EAR file for which you want to configure new resource adapters. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new resource adapters from the tree.
4. Expand the JAR file to view its contents.
5. Right-click the **Resource Adapters** folder, and click **New**.
6. Configure the resource adapter settings in the resulting property dialog.
7. Click **OK**.
8. Click **File** > **Save** on the menu bar to save your changes.

clientRAR tool

This topic describes the command line syntax for the client resource adapter installation tool.

If this tool is used to add or delete resource adapters on the server, then only the client can use the resource adapter. If the resource adapter is installed on the server using the wsadmin tool or the administrative console, then do not use the clientRAR tool remove it. Only resource adapters that are installed using the clientRAR tool should be removed using the clientRAR tool.

The command line invocation syntax for the `clientRAR` tool follows:

```
clientRAR [-help | -?] [-CRDcom.ibm.ws.client.installedConnectors=<dir>] <task> <archive>

where
-help, -?
Print the usage information.
-CRDcom.ibm.ws.client.installedConnectors
The directory where resource adapters are installed.
This will override the system property of the same name
(com.ibm.ws.client.installedConnectors).


<task>
The task to perform: add - install, delete - uninstall.

<archive>
if task=add then this is the fully qualified name of the resource adapter archive file.
If task=delete then this is the filename of the resource adapter archive to be uninstalled.
```

The following examples demonstrate correct syntax.

On the Windows operating systems:

- clientRAR add c:\rars\myrar.rar
- clientRAR delete myrar.rar

On the UNIX operating systems:

- ./clientRAR add /usr/rars/myrar.rar
- ./clientRAR delete myrar.rar

Configuring new connection factories for resource adapters

Use the Application Client Resource Configuration Tool (ACRCT) to configure new connection factories for resource adapters.

Complete this task to configure new connection factories for resource adapters.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure new connection factories. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new connection factories from the tree.

4. Expand the JAR file to view its contents.
5. Click the **Resource Adapters** folder.
6. Expand the resource adapter for which you want to create connection factories.
7. Right-click the **Connection Factories** folder and click **New**.
8. Configure the connection factory properties in the resulting property dialog.
9. Click **OK**.
10. Click **File** > **Save** on the menu bar to save your changes.

Resource adapter connection factory settings:

Use this panel to view or change the configuration properties of the selected resource adapter connection factory.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Adapters**. Right-click the **Connection Factories** folder, and click **New**. The following fields appear on the **General** tab.

Name:

The name by which this connection factory is known for administrative purposes within WebSphere Application Server. The name must be unique within the resource adapter connection factories across the product administrative domain.

Data type String

Description:

An optional description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

The JNDI name that is used to match this resource adapter connection factory definition to the deployment descriptor. This entry should be a `resource-ref` name.

Data type String

User Name:

The **User Name** used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly when getting a connection. If this field is used, then the Properties field `UserName` is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User Name** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly when getting a connection.

Data type String

Password:

Specifies an encrypted password. If you complete this field, then the **Password** field in the Properties box is ignored.

If you specify a value for the **UserName** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Type:

A drop-down list of all the `connectionFactoryInterfaces` as defined for the factories in the Resource Adapter Archive.

For each **Type**, there is a set of properties specified in the Properties box. This set of properties is constructed by retrieving the properties from each connection definition object. For any existing connection factories that are displayed for updating, this list of properties is overlaid with the properties specified for the objects. When the **Type** field is changed, the properties also change to reflect the correct properties for that type.

Data type String

Configuring administered objects

Before you configure new administered objects, you must complete the following prerequisites:

1. Install the Resource Adapter Archive file (RAR) using the clientRAR tool.
2. Configure the resource adapter for the `.ear` file, using the Application Client Resource Configuration Tool (ACRCT) tool.

Complete this task to configure new administered objects for installed resource adapters.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure new administered objects. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new administered objects from the tree.
4. Expand the JAR file to view its contents.
5. Click the **Resource Adapters** folder.
6. Expand the resource adapter for which you want to create administered objects.
7. Right-click the **Administered Objects** folder and click **New**.
8. Configure the administered object properties in the resulting property dialog.
9. Click **OK**.
10. Click **File** > **Save** on the menu bar to save your changes.

Administered objects settings:

Use this panel to view or change the configuration properties of the selected administered objects.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Adapters** > *resource_adapter_instance*. Right-click **Administered Objects** and click **New**. The following fields appear on the **General** tab.

The settings for administered objects are handled similarly to connection factories. When updating administered objects, use the same panels that you used to create administered objects.

Name:

The name by which this administered object is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the resource adapter administered objects across the product administrative domain.

Data type String

Description:

An optional description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

This entry is a `resource-env-ref` name, a `message-destination-ref` name (if the `message-destination-ref` has no link), or a `message-destination` link.

Data type String

Type:

A drop-down list of all the administered object class-interface pairs as defined for the admin objects in the Resource Adapter Archive (RAR) file.

For each **Type**, there is a set of properties specified in the Properties box. This set of properties is constructed by retrieving the properties from each administered object definition. For any existing administered objects that are displayed for updating, this list of properties is overlaid with the properties specified for the objects. When the **Type** field is changed, the properties also change to reflect the correct properties for that type.

Data type String

Resource adapter settings

Use this panel to view or change the configuration properties of the resource adapter. These configuration properties control how resource adapters are created.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Adapter**. Right-click **Resource Adapter** and click **New**. The following fields appear on the **General** tab.

Name
The name by which this Resource Adapter is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the Resource Adapters across the product administrative domain.

Data type String

Description
A description of this resource adapter for administrative purposes within IBM WebSphere Application Server.

Data type String

Class Path
Any additional class path. The path to the resource adapter directory is automatically added.

Data type String
Default The path to your Resource Adapter directory.

Native Path
The native path where the Resource Adapter is located. Enter any additional native class path here.

Data type String

Resource Adapter Name
A mandatory field that points to an installed resource adapter subdirectory. The entry does not represent the full directory name for the resource adapter. The full directory name is the installed resource adapter path, plus the resource adapter name.

Data type String

Installed Resource Adapter Path
The directory where resource adapters are installed. If you do not complete this field, then the default takes effect.

If you specify the value, `${CONNECTOR_INSTALL_ROOT}`, then this value replaces the value of the `CLIENT_CONNECTOR_INSTALL_ROOT` variable on the machine on which the client application runs. This action allows the application to run easily on different machines, where the client installation might be in different locations.

Data type String
Default ${CONNECTOR_INSTALL_ROOT}

Starting the Application Client Resource Configuration Tool and opening an EAR file
You can perform many tasks by starting the Application Client Resource Configuration Tool (ACRCT). Many of these tasks also involve then opening an EAR file.

Note: This task only applies to J2EE application clients.

Use these steps to start the Application Client Resource Configuration Tool. When you start the tool, one of the most common tasks that you perform is opening and modifying the components of EAR files.

1. Open a command prompt and change to the *app_server_root*\bin directory.
2. Run the clientConfig.bat file for a Windows system or the clientConfig.sh file for a UNIX system.
3. Open an EAR file within the Application Client Resource Configuration Tool (ACRCT):
   - Click **File** > **Open**.
   - Select the file and click **Open**.
4. Save your changes to the file and close the tool:
   - Click **File** > **Save**.
   - Click **File** > **Exit**.

Data sources for the Application Client

WebSphere Application Server and the Application Client for WebSphere Application Server do not provide client database drivers to be used directly from a J2EE application client. If your application client accesses a database directly, you must provide the database drivers on the client machine.

You can contact your database vendor to acquire client database driver code and licenses. In addition, data sources configured on the server and looked up on the client do not participate in global transactions. Instead of accessing the database directly, it is recommended that your client application use an enterprise bean. Accessing a database through an enterprise bean eliminates the need to have database drivers on the client machine, since the database access is handled by the enterprise bean running on WebSphere Application Server. For a current list of providers that are supported on WebSphere Application Server visit the Supported hardware, software, and APIs Web site:

Data source properties for application clients

Use this page to create or modify the data sources.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Data Source Providers** > *Data source provider instance*. Right-click **Data Sources** and click **New**. The following fields are displayed on the **General** tab:

Name

Specifies the display name of this data source.

Data type String

Description

Specifies a text description of the data source.

Data type String

JNDI Name

The application client run time uses this field to retrieve configuration information.

Database Name

The name of the database to which you want to connect.

User

Use the user ID with the Password property, for authentication if the calling application does not provide a user ID and password explicitly.

If you specify a value for the User ID property, then you must also specify a value for the Password property. The connection factory User ID and Password properties are used if the calling application does not provide a user ID and password explicitly.

Password
Use the password with the User ID property, for authentication if the calling application does not provide a user ID and password explicitly.

If you specify a value for the Password property, then you must also specify a value for the User ID property.

Re-Enter Password
Confirms the password.

Custom Properties
Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new data source providers (JDBC providers) for application clients
You can create new data source providers, also known as JDBC providers, for your application client using the Application Client Resource Configuration Tool (ACRCT) .

During this task, you create new data source providers, also known as JDBC providers, for your application client. In a separate administrative task, install the Java code for the required data source provider on the client machine on which the application client resides.

Use this task to connect application clients to relational databases.

1. Start the Application Client Resource Configuration Tool (ACRCT) and open the EAR file for which you want to configure the new data source provider. The EAR file contents display in a tree view.
2. Select the JAR file in which you want to configure the new data source provider from the tree.
3. Expand the JAR file to view its contents.
4. Click the **Data Source Providers** folder. Do one of the following:
   - Right-click the folder and click **New Provider**.
   - Click **Edit > New** on the menu bar.
5. Configure the data source provider properties in the resulting property dialog.
6. Click **OK** when you finish.
7. Click **File > Save** on the menu bar to save your changes.

Example: Configuring data source provider and data source settings
You can configure data source provider and data source settings.

The purpose of this article is to help you to configure data source provider and data source settings.
- Required fields:
  - Data Source Provider Properties page: name
  - Data Source Properties page: name, jndiName
- Special cases:
  - The user name and password fields have no equivalent XMI tags. You must specify these fields in the custom properties.
  - The password is encrypted when you use the Application Client Resource Configuration Tool (ACRCT). If you do not use the ACRCT the field cannot be encrypted.
- Example:

```
<resources.jdbc:JDBCProvider xmi:id="JDBCProvider_1" name="jdbcProvider:name"
description="jdbcProvider:description" implementationClassName="jdbcProvider:
ImplementationClass">
<classpath>jdbcProvider:classPath</classpath>
<factories xmi:type="resources.jdbc:WAS40DataSource" xmi:id="WAS40DataSource_1"
name="jdbcFactory:name" jndiName="jdbcFactory:jndiName"
description="jdbcFactory:description" databaseName="jdbcFactory:databasename">
<propertySet xmi:id="J2EEResourcePropertySet_13">
<resourceProperties xmi:id="J2EEResourceProperty_13" name="jdbcFactory:customName"
value="jdbcFactory:customValue"/>
<resourceProperties xmi:id="J2EEResourceProperty_14" name="user"
value="jdbcFactory:user"/>
<resourceProperties xmi:id="J2EEResourceProperty_15" name="password"
value="{xor}NTs9PBk+PCswLSZlMT4yOg=="/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_14">
<resourceProperties xmi:id="J2EEResourceProperty_16" name="jdbcProvider:customName"
value="jdbcProvider:customeValue"/>
</propertySet>
</resources.jdbc:JDBCProvider>
```

Data source provider settings for application clients

Use this page to create a data source under a JDBC provider which provides the specific JDBC driver implementation class.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **Data Source Providers** > and click **New**. The following fields appear on the **General** tab:

Name:

Specifies the display name for the data source.

For example you can set this field to *Test Data Source*.

Data type String

Description:

Specifies a text description for the resource.

Data type String

Class Path:

A list of paths or `.jar` file names which together form the location for the resource provider classes.

Implementation class:

Use this setting to perform database specific functions.

Data type String
Default Dependent on JDBC driver implementation class

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new data sources for application clients

During this task, you create new data sources for your application client.

1. Click the data source provider for which you want to create a data source in the tree. Take one of the following actions as needed:
   - Configure a new data source provider.
   - Click an existing data source provider.
2. Expand the data source provider to view its **Data Sources** folder.
3. Click the data source folder. Take one of the following actions as needed:
   - Right click the data source folder and click **New Factory**.
   - Click **Edit > New** on the menu bar.
4. Configure the data source properties in the displayed fields.
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring mail providers and sessions for application clients

You can edit the configurations of JavaMail sessions and providers for your application clients using the Application Client Resource Configuration Tool (ACRCT).

Use the Application Client Resource Configuration Tool (ACRCT) to edit the configurations of JavaMail sessions and providers for your application clients to use.

1. Start the ACRCT.
2. Open an EAR file.
3. Locate the JavaMail objects in the tree that displays. For example, if your file contains JavaMail sessions, expand **Resources** > ***application*.jar** > **Mail Providers** > ***java_mail_provider_instance*** > **Mail Sessions**.

   In this example, ***java_mail_provider_instance*** is a particular JavaMail provider.

The JavaMail session instances are located in the **JavaMail Sessions** folder.

Mail provider settings for application clients

Use this page to implement the JavaMail API and create mail sessions.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **Mail Providers** > and click **New**. The following fields appear on the **General** tab:

Name:

The name of the JavaMail resource provider.

Description:

An optional description for the resource provider.

Class Path:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

Protocol:

Specifies the name of the protocol.

Classname:

Specifies the name of the class implementing the protocol. Leave this field blank if you want to use the default implementation.

Type:

This menu contains the following two values: `TRANSPORT` or `STORE`.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Mail session settings for application clients

Use this page to configure mail session properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Mail Providers** > *mail provider instance*. Right-click **Mail Sessions** and click **New**. The following fields appear on the **General** tab:

Name:

Represents the administrative name of the JavaMail session object.

Description:

Provides an optional description for your administrative records.

JNDI Name:

The application client run time uses this field to retrieve configuration information.

Mail Transport Host:

Specifies the server to connect to when sending mail.

Mail Transport Protocol:

Specifies the transport protocol to use when sending mail.

Mail Transport User:

Specifies the user ID to use when the mail transport host requires authentication.

Mail Transport Password:

Specifies the password to use when the mail transport host requires authentication.

Enable strict Internet address parsing:

Specifies whether the recipient addresses must be parsed strictly in compliance with RFC 822, which is a specifications document issued by the Internet Architecture Board.

This setting is not generally used for most mail applications. RFC 822 syntax for parsing addresses effectively enforces a strict definition of a valid e-mail address. If you select this setting, JavaMail will adhere to RFC 822 syntax and reject recipient addresses that do not parse into valid e-mail addresses (as defined by the specification). If you do not select this setting, JavaMail will not adhere to RFC 822 syntax and will accept recipient addresses that do not comply with the specification. By default, this setting is deselected. You can view the RFC 822 specification at the following URL for the World Wide Web Consortium (W3C): http://www.w3.org/Protocols/rfc822/.

Re-Enter Password:

Confirms the password.

Mail From:

Specifies the mail originator.

Mail Store Host:

Specifies the mail account host (or ″domain″) name.

Mail Store User:

Specifies the user ID of the mail account.

Mail Store Password:

Specifies the password of the mail account.

Re-Enter Password:

Confirms the password.

Mail Store Protocol:

Specifies the protocol to be used when receiving mail.

Mail Debug:

When `true`, JavaMail interaction with mail servers, along with these mail session properties are printed to the `stdout` file.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring JavaMail provider and JavaMail session settings for application clients

You can configure JavaMail provider and JavaMail session settings. This topic provides the required fields, special cases, and an example.

The purpose of this article is to help you configure JavaMail provider and JavaMail session settings.

- Required fields:
  - JavaMail Provider Properties page: name, and at least one protocol provider
  - JavaMail Session Properties page: name, jndiName, mail transport protocol, mail store protocol
- Special cases:
  - The password is encrypted when using the ACRCT tool. Without the tool, you cannot encrypt this field.
- Example:

```
<resources.mail:MailProvider xmi:id="MailProvider_1" name="Default Mail Provider"
description="IBM JavaMail Implementation">
<classpath>mailProvider:classpath</classpath>
<factories xmi:type="resources.mail:MailSession" xmi:id="MailSession_1"
name="mailSession:name" jndiName="mailSession:jndiName"
description="mailSession:description" mailTransportHost="mailSession:mailTransportHost"
mailTransportUser="mailSession:mailTransportUser"
mailTransportPassword="{xor}Mj42Mww6LCw2MDFlMT4yOg=="
mailFrom="mailSession:mailFrom" mailStoreHost="mailSession:mailStoreHost"
mailStoreUser="mailSession:mailStoreUser"
mailStorePassword="{xor}Mj42Mww6LCw2MDFlMT4yOg==" debug="true"
mailTransportProtocol="ProtocolProvider_1" mailStoreProvider="ProtocolProvider_1">
<propertySet xmi:id="J2EEResourcePropertySet_1">
<resourceProperties xmi:id="J2EEResourceProperty_1"
name="mailSession:customName" value="mailSession:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_2">
<resourceProperties xmi:id="J2EEResourceProperty_2" name="mailProvider:customName"
value="mailProvider:customValue"/>
</propertySet>
<protocolProviders xmi:id="ProtocolProvider_1" protocol="smtp"
classname="smtp:className"/>
<protocolProviders xmi:id="ProtocolProvider_2" protocol="pop3"
classname="pop3:className"/>
<protocolProviders xmi:id="ProtocolProvider_3" protocol="imap"
classname="imap:className"/>
</resources.mail:MailProvider>
```

Configuring new mail sessions for application clients

You can use the Application Client Resource Configuration Tool (ACRCT) to configure new mail sessions for your application client.

During this task, you configure new mail sessions for your application client. The mail sessions are associated with the pre-configured default mail provider supplied by the product.

1. Start the Application Client Resource Configuration Tool (ACRCT) and open the EAR file. The EAR file contents are displayed in a tree view.
2. Select the JAR file in which you want to configure the new JavaMail session.
3. Expand the JAR file to view its contents.
4. Click **Mail Providers** > **Mail Provider** > **Mail Sessions**. Complete one of the following actions:
   - Right click the **Mail Sessions** folder and select **New Factory**.
   - Click **Edit** > **New** on the menu bar.
5. Configure the Mail Session properties in the displayed fields.
6. Click **OK**.
7. Click **File** > **Save** on the menu bar to save your changes.

URLs for application clients

A *Uniform Resource Locator* (URL) is an identifier that points to an electronically accessible resource, such as a directory file on a machine in a network, or a document stored in a database.

URLs appear in the format *scheme*:*scheme_information*.

You can represent a *scheme* as `http`, `ftp`, `file`, or another term that identifies the type of resource and the mechanism by which you can access the resource.

In a World Wide Web browser location or address box, a URL for a file available using HyperText Transfer Protocol (HTTP) starts with `http:`. An example is `http://www.ibm.com`. Files available using File Transfer Protocol (FTP) start with `ftp:`. Files available locally start with `file:`.

The *scheme_information* commonly identifies the Internet machine making a resource available, the path to that resource, and the resource name. The scheme_information for HTTP, FTP and File generally starts with two slashes (//), then provides the Internet address separated from the resource path name with one slash (/). For example,

`http://www-4.ibm.com/software/webservers/appserv/library.html`.

For HTTP and FTP, the path name ends in a slash when the URL points to a directory. In such cases, the server generally returns the default index for the directory.

URL providers for the Application Client Resource Configuration Tool

A URL provider implements the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP). This provider, comprised of a pair of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Configuring new URL providers for application clients

You can create URL providers and URLs for your client application using the Application Client Resource Configuration Tool (ACRCT).

During this task, you create URL providers and URLs for your client application. In a separate administrative task, you must install the Java code for the required URL provider on the client machine on which the client application resides.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new URL provider. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new URL provider from the tree.
4. Expand the JAR file to view the contents.
5. Click the folder called **URL Providers**. Complete one of the following actions:
   - Right click the folder and select **New**.
   - Click **Edit** > **New** on the menu bar.
6. Configure the URL provider properties in the resulting property dialog.
7. Click **OK**.
8. Click **File** > **Save** on the menu bar to save your changes.

Configuring URL providers and sessions using the Application Client Resource Configuration Tool

You can edit the configurations of URL providers and URLs to be used by your application clients using the Application Client Resource Configuration Tool (ACRCT).

Use the Application Client Resource Configuration Tool (ACRCT) to edit the configurations of URL providers and URLs to be used by your application clients.

1. Start the ACRCT.
2. Open an EAR file.
3. Locate the URL objects in the tree that displays. For example, if your file contains URL providers and URLs, expand **Resources** -> ***application*.jar** -> **URL Providers** -> ***url_provider_instance***

   where ***url_provider_instance*** is a particular URL provider.
4. If you expand the tree further, you will also see the **URLs** folders containing the URL instances for each URL provider instance.

URL settings for application clients:

Use this page to implement the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP).

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **URL Providers** > *URL provider instance*. Right-click **URLs** and click **New**. The following fields appear on the **General** tab.

This provider, comprised of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Name:

The administrative name for the URL.

Description:

This is an optional description of the URL for your administrative records.

JNDI Name:

The application client run time uses this field to retrieve configuration information.

URL:

A Uniform Resource Locator (URL) name that points to an Internet or intranet resource. For example: `http://www.ibm.com`.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

URL provider settings for application clients:

Use this page create new URL providers.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right click **URL Providers**, and click **New**. The following fields appear on the **General** tab.

A URL provider implements the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP). This provider, comprised of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Name:

Administrative name for the URL.

Description:

Optional description of the URL, for your administrative records.

Class Path:

A list of paths or JAR file names which together form the location for the resource provider classes.

Protocol:

Protocol supported by this stream handler. For example, `nntp`, `smtp`, `ftp`, and so on.

To use the default protocol, leave this field blank.

Stream handler class:

Fully qualified name of a User-defined Java class that extends the `java.net.URLStreamHandler` for a particular URL protocol, such as FTP.

To use the default stream handler, leave this field blank.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring URL and URL provider settings for application clients

You can configure URL and URL provider settings. This topic provides the required fields and an example.

The purpose of this article is to help you to configure URL and URL provider settings.

- Required fields:
  - URL Properties page: name, jndiName, url
  - URL Provider Properties page: name
- Example:

```
<resources.url:URLProvider xmi:id="URLProvider_1" name="urlProvider:name"
description="urlProvider:description"
streamHandlerClassName="urlProvider:streamHandlerClass"
protocol="urlProvider:protocol">
<classpath>urlProvider:classpath</classpath>
<factories xmi:type="resources.url:URL" xmi:id="URL_1" name="urlFactory:name"
jndiName="urlFactory:jndiName" description="urlFactory:description"
spec="urlFactory:url">
<propertySet xmi:id="J2EEResourcePropertySet_18">
<resourceProperties xmi:id="J2EEResourceProperty_20" name="urlFactory:customName"
value="urlFactory:customValue"/>
</propertySet>
```

```
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_19">
<resourceProperties xmi:id="J2EEResourceProperty_21" name="urlProvider:customName"
value="urlProvider:customValue"/>
</propertySet>
</resources.url:URLProvider>
```

Configuring new URLs with the Application Client Resource Configuration Tool

You can use URLs for your client application using the Application Client Resource Configuration Tool (ACRCT).

During this task, you create URLs for your client application.

1. Click the URL provider for which you want to create a URL in the tree. Complete one of the following:
   - Configure a new URL provider.
   - Click an existing URL provider.
2. Expand the URL provider to view the **URLs** folder.
3. Click the URL folder. Complete one of the following actions:
   - Right click the folder and click **New**.
   - Click **Edit -> New** on the menu bar.
4. Configure the URL properties in the displayed fields.
5. Click **OK** when you finish.
6. Click **File** > **Save** in the menu bar to save your changes.

Asynchronous messaging in WebSphere Application Server using JMS

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The JMS interface provides a common way for Java programs (clients and J2EE applications) to create, send, receive, and read asynchronous requests as JMS messages.

This topic provides a generic overview of asynchronous messaging using the JMS support provided by WebSphere Application Server.

The base support for asynchronous messaging using the JMS API provides the common set of JMS interfaces and associated semantics that define how a JMS client can access the facilities of a JMS provider. This support enables WebSphere product J2EE applications, as JMS clients, to exchange messages asynchronously with other JMS clients, by using JMS destinations (queues or topics). A J2EE application can use JMS queue destinations for point-to-point messaging and JMS topic destinations for Publisher and Subscriber messaging. A J2EE application can explicitly poll for messages on a destination, and then retrieve messages for processing by business logic beans (enterprise beans).

With the base JMS and XA support, the J2EE application uses standard JMS calls to process messages, including any responses or outbound messaging. An enterprise bean can handle responses acting as a sender bean, or within the enterprise bean that receives the incoming messages. Optionally, this process can use two-phase commit within the scope of a transaction. This level of function for asynchronous messaging is called *bean-managed messaging*, and gives an enterprise bean complete control over the messaging infrastructure, for example, connection and session pool management. The common container has no role in bean-managed messaging.

WebSphere Application Server also supports automatic asynchronous messaging using message-driven beans (a type of enterprise bean defined in the EJB 2.0 specification) and JMS listeners (part of the JMS application server facilities). Messages are automatically retrieved from JMS destinations, optionally within a transaction, then sent to the message-driven bean in a J2EE application, without the application having to explicitly poll JMS destinations.

Java Message Service (JMS) providers for clients

This topic describes the different ways that client applications can use JMS providers with WebSphere Application Server. A JMS provider enables use of the Java Message Service (JMS) and other message resources in WebSphere Application Server.

IBM WebSphere Application Server supports asynchronous messaging through the use of a JMS provider and its related messaging system. JMS providers must conform to the JMS specification version 1.1. To use message-driven beans the JMS provider must support the optional Application Server Facility (ASF) function defined within that specification, or support an inbound resource adapter as defined in the JCA specification version 1.5.

The service integration technologies of IBM WebSphere Application Server can act as a messaging system when you have configured a service integration bus that is accessed through the default messaging provider. This support is installed as part of WebSphere Application Server, administered through the administrative console, and is fully integrated with the WebSphere Application Server runtime.

WebSphere Application Server also includes support for the following JMS providers:

WebSphere MQ
Provided for use with supported versions of WebSphere MQ.

Generic
Provided for use with any 3rd party messaging system which supports ASF.

For backwards compatibility with earlier releases, WebSphere Application Server also includes support for the V5 default messaging provider which enables you to configure resources for use with the WebSphere Application Server version 5 Embedded Messaging system. The V5 default messaging provider can also be used with a service integration bus.

WebSphere applications can use messaging resources provided by any of these JMS providers. However the choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you may already have a messaging infrastructure based on WebSphere MQ. In this case you may either connect directly using the included support for WebSphere MQ as a JMS provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

The service integration bus also provides access to a default messaging provider. This is a J2EE 1.4 compliant JMS messaging provider which is fully integrated with WebSphere Application Server. You can use it in multiple server configurations for messaging interactions with a WebSphere MQ network.

Configuring Java messaging client resources

To configure Java messaging client resources, you create new JMS provider configurations for your application client. The application client can use a messaging service through the Java Message Service APIs. A JMS provider provides two kinds of J2EE factories. One is a *JMS connection factory*, and the other is a *JMS destination factory*.

In a separate administrative task, install the Java Message Service (JMS) client on the client machine where the application client resides. The messaging product vendor must provide an implementation of the JMS client. For more information, see your messaging product documentation.

Note: When completing this task, you can either create a new messaging provider, or you can use an existing one.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new JMS provider. The EAR file contents are in the displayed tree view.

3. Select the JAR file in which you want to configure the new JMS provider from the tree.
4. Expand the JAR file to view its contents.
5. Optionally right-click **Messaging Providers** and select **New**, if you want to create and use a new messaging provider.
6. Configure the JMS provider properties in the resulting property dialog.
7. Click **OK**.
8. Click **File** > **Save**.

Configuring new JMS providers with the Application Client Resource Configuration Tool

You can create new Java Message Service (JMS) provider configurations for the Application Client. The Application Client makes use of a messaging service through the JMS interfaces.

During this task, you create new Java Message Service (JMS) provider configurations for the Application Client. The Application Client makes use of a messaging service through the JMS interfaces. A JMS provider provides two kinds of J2EE resources. One is a JMS connection factory, and the other is a JMS destination.

In a separate administrative task, you must install the JMS client on the client machine where your particular application client resides. The messaging product vendor must provide an implementation of the JMS client. For more information, see your messaging product documentation.

1. Start the Application Client Resource Configuration Tool and open the EAR file for which you want to configure the new JMS provider. The EAR file contents are displayed in a tree view.
2. From the tree, select the JAR file in which you want to configure the new JMS provider.
3. Expand the JAR file to view its contents.
4. Right-click **Messaging Providers**. Complete one of the following actions:
   - Right click the folder and select **New**.
   - On the menu bar, click **Edit > New**.
5. In the resulting property dialog, configure the JMS provider properties.
6. Click **OK** when finished.
7. Click **File -> Save** on the menu bar to save your changes.

JMS provider settings for application clients

Use this page to configure properties of the Java Message Service (JMS) provider, if you want to use a JMS provider other than the default messaging provider or the WebSphere MQ as a JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right click **Messaging Providers**, and click **New**. The following fields appear on the **General** tab.

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String

Description:

A description of the JMS provider, for administrative purposes.

Data type String

Class Path:

A list of paths or `.jar` file names which together form the location for the resource provider classes.

Context factory class:

The Java class name of the initial context factory for the JMS provider.

For example, for an LDAP service provider the value has the form: `com.sun.jndi.ldap.LdapCtxFactory`.

Data type String

Provider URL:

The JMS provider URL for external JNDI lookups.

For example, an LDAP URL for a JMS provider has the form: `ldap://hostname.company.com/contextName`.

Data type String

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Default Provider connection factory settings

Use this panel to view or change the configuration properties of the selected JMS connection factory for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server. These configuration properties control how connections are created between the JMS provider and the service integration bus that it uses

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Connection Factories** and click **New**. The following fields appear on the **General** tab.

Settings that have a default value display the appropriate value. Any settings that have fixed values have a drop down menu.

Name:

The name of the connection factory.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

The JNDI name that is used to match this Resource Adapter connection factory definition to the deployment descriptor. This entry is a `resource-ref` name.

Data type String

User Name:

The **User Name** used with the **Password** property for connecting to an application.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly. If a user name and password are specified, then an authentication alias is created for the factory where the password is encrypted.

Data type String

Password:

The password used to authenticate connection to an application.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Bus Name:

The name of the bus to which the connection factory connects.

Data type String

Client Identifier:

The name of the client. Required for durable topic subscriptions.

Data type String

Nonpersistent Messaging Reliability:

The reliability applied to nonpersistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus** destination. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default
ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Persistent Message Reliability:

The reliability applied to persistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus destination**. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default
ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Durable Subscription Home:

The name of the durable subscription home.

Data type String

Share durable subscriptions:

Controls whether or not durable subscriptions are shared across connections with members of a server cluster.

Normally, only one session at a time can have a TopicSubscriber for a particular durable subscription. This property enables you to override this behavior, to enable a durable subscription to have multiple simultaneous consumers.

Data type Selection list
Default In cluster
Range

In cluster
Allows sharing of durable subscriptions when connections are made from within a server cluster.

Always shared
Durable subscriptions can be shared across connections.

Never shared
Durable subscriptions are never shared across connections.

Read Ahead:

Controls the read-ahead optimization during message delivery.

Default Default
Range Default, AlwaysOn and AlwaysOff

Target:

The name of the Workload Manager target group containing the messaging engine.

Data type String

Target Type:

The type of Workload Manager target group that contains the messaging engine.

Default BusMember
Range BusMember, Custom, ME

Target Significance:

The priority of significance for the target specified.

Default Preferred
Range Preferred, Required

Target Inbound Transport Chain:

The name of the protocol that resolves to a group of messaging engines.

Data type String

Provider Endpoints:

The list of comma separated endpoints used to connect to a bootstrap server.

Type a comma-separated list of endpoint triplets with the syntax: host:port:protocol.

Example `merlin:7276:BootstrapBasicMessaging,Gandalf:5557:BootstrapSecureMessaging`

where

`BootstrapBasicMessaging` corresponds to the remote protocol InboundBasicMessaging (JFAP-TCP/IP).

Default

- If the host name is not specified, then the default `localhost` is used as a default value.
- If the port number is not specified, then `7276` is used as a default value.
- If the chain name is not specified, a predefined chain, such as `BootstrapBasicMessaging`, is used as a default value.

Connection Proximity:

The proximity that the messaging engine should have to the requester.

Default	Bus
Range	Bus, Host, Cluster, Server

Temporary Queue Name Prefix:

The prefix to apply to the names of temporary queues. This name is a maximum of 12 characters.

Data type	String

Temporary Topic Name Prefix:

The prefix to apply to the names of temporary topics. This name is a maximum of 12 characters.

Data type	String

Default Provider queue connection factory settings

Use this panel to view or change the configuration properties of the selected JMS queue connection factory for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server. These configuration properties control how connections are created between the JMS provider and the service integration bus that it uses

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Queue Connection Factories** and click **New**. The following fields appear on the **General** tab.

Settings that have a default value display the appropriate value. Any settings that have fixed values have a drop down menu.

Name:

The name of the queue connection factory.

Data type	String

Description:

A description of this queue connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String

JNDI Name:

The JNDI name that is used to match this queue connection factory definition to the deployment descriptor. This entry is a `resource-ref` name.

Data type	String

User Name:

The **User Name** used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly. If this field is used, then the Properties field `UserName` is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User Name** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly. If a user name and password are specified, then an authentication alias is created for the factory where the password is encrypted.

Data type String

Password:

The password used to create an encrypted. If you complete this field, then the Password field in the Properties box is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Bus Name:

The name of the bus to which the queue connection factory connects.

Data type String

Client Identifier:

The client identifier. Required for durable topic subscriptions.

Data type String

Nonpersistent Messaging Reliability:

The reliability applied to nonpersistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus** destination. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Persistent Message Reliability:

The reliability applied to persistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus destination**. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Read Ahead:

Controls the read-ahead optimization during message delivery.

Default Default
Range Default, AlwaysOn and AlwaysOff

Target:

The name of the Workload Manager target group containing the messaging engine.

Data type String

Target Type:

The type of Workload Manager target group that contains the messaging engine.

Default BusMember
Range BusMember, Custom, Destination, ME

Target Significance:

The priority of significance for the target specified.

Default Preferred
Range Preferred, Required

Target Inbound Transport Chain:

The name of the protocol that resolves to a group of messaging engines.

Data type String

Provider Endpoints:

The list of comma separated endpoints used to connect to a bootstrap server.

Type a comma-separated list of endpoint triplets with the syntax: host:port:protocol.

Example `localhost:7777:BootstrapBasicMessaging`

where

`BootstrapBasicMessaging` corresponds to the remote protocol InboundBasicMessaging (JFAP-TCP/IP).

Default

- If the host name is not specified, then the default `localhost` is used as a default value.
- If the port number is not specified, then `7276` is used as a default value.
- If the chain name is not specified, a predefined chain, such as `BootstrapBasicMessaging`, is used as a default value.

Connection Proximity:

The proximity that the messaging engine should have to the requester.

Default Bus, Cluster, Server
Range Bus, Host

Temporary Queue Name Prefix:

The prefix to apply to the names of temporary queues. This name is a maximum of 12 characters.

Data type String

Default Provider topic connection factory settings

Use this panel to view or change the configuration properties of the selected JMS topic connection factory for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server. These configuration properties control how connections are created between the JMS provider and the service integration bus that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Topic Connection Factories** and click **New**. The following fields appear on the **General** tab.

Settings that have a default value display that appropriate value. Any settings that have fixed values have a drop down menu.

Name:

The name of the topic connection factory.

Data type String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

The JNDI name that is used to match this topic connection factory definition to the deployment descriptor. This entry is a `resource-ref` name.

Data type String

User Name:

The **User Name** used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly. If this field is used, then the Properties field `UserName` is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User Name** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly. If a user name and password are specified, then an authentication alias is created for the factory where the password is encrypted.

Data type String

Password:

The password used to create an encrypted. If you complete this field, then the Password field in the Properties box is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Bus Name:

The name of the bus to which the topic connection factory connects.

Data type String

Client Identifier:

The name of the client. This field is required for durable topic subscriptions.

Data type String

Nonpersistent Messaging Reliability:

The reliability applied to nonpersistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus** destination. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Persistent Message Reliability:

The reliability applied to persistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus destination**. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Durable Subscription Home:

The name of the durable subscription home.

Data type String

Share durable subscriptions:

Controls whether or not durable subscriptions are shared across connections with members of a server cluster.

Normally, only one session at a time can have a TopicSubscriber for a particular durable subscription. This property enables you to override this behavior, to enable a durable subscription to have multiple simultaneous consumers.

Data type Selection list
Default In cluster
Range

In cluster
Allows sharing of durable subscriptions when connections are made from within a server cluster.

Always shared
Durable subscriptions can be shared across connections.

Never shared
Durable subscriptions are never shared across connections.

Read Ahead:

Controls the read-ahead optimization during message delivery.

Default	Default
Range	Default, AlwaysOn and AlwaysOff

Target:

The name of the Workload Manager target group containing the messaging engine.

Data type	String

Target Type:

The type of Workload Manager target group that contains the messaging engine.

Default	BusMember
Range	BusMember, Custom, ME

Target Significance:

The priority of significance for the target specified.

Default	Preferred
Range	Preferred, Required

Target Inbound Transport Chain:

The name of the protocol that resolves to a group of messaging engines.

Data type	String

Provider Endpoints:

The list of comma separated endpoints used to connect to a bootstrap server.

Type a comma-separated list of endpoint triplets with the syntax: host:port:protocol.

Example	`localhost:7777:BootstrapBasicMessaging` where `BootstrapBasicMessaging` corresponds to the remote protocol InboundBasicMessaging (JFAP-TCP/IP).
Default	• If the host name is not specified, then the default `localhost` is used as a default value. • If the port number is not specified, then `7276` is used as a default value. • If the chain name is not specified, a predefined chain, such as `BootstrapBasicMessaging`, is used as a default value.

Connection Proximity:

The proximity that the messaging engine should have to the requester.

Default	Bus
Range	Bus, Host, Cluster, Server

Temporary Topic Name Prefix:

The prefix to apply to the names of temporary topics. This name is a maximum of 12 characters.

Data type	String

Default Provider queue destination settings

Use this panel to view or change the configuration properties of the selected JMS queue destination for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Queue Destinations**. Click **New**. The following fields appear on the **General** tab.

Name:

The name of the queue destination factory. You must complete this field.

Data type	String

Description:

A description of this queue destination for administrative purposes within WebSphere Application Server.

Data type	String

JNDI Name:

The JNDI name used to match this definition to a deployment descriptor `resource-env-ref` name.

Data type	String

Queue Name:

The name of the queue.

Data type	String

Delivery Mode:

The delivery mode for messages sent to this destination.

Data type	String
Range	Application, Persistent or NonPersistent
Default	Application

Time to Live:

The default length of time from its dispatch time that a message sent to this destination should be retained by the system, where **0** indicates that time to live value does not expire. Value from the producer is used if the Time to Live field is not completed.

Data type	Integer
Units	Milliseconds

Priority:

The priority for messages sent to this destination. The value from the producer is used if not completed.

Data type	Integer
Range	0 to 9 with **0** as the lowest priority and **9** as the highest priority

Read Ahead:

Used to control read-ahead optimization during message delivery.

Data type	String
Range	AsConnection, AlwaysOn and AlwaysOff
Default	AsConnection

Default Provider topic destination settings

Use this panel to view or change the configuration properties of the selected JMS topic destination for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Topic Destinations**, and click **New**. The following fields appear on the **General** tab.

Name:

The name of the topic destination entry.

Data type	String

Description:

A description of the entry.

Data type	String

JNDI Name:

The JNDI name used to match this definition to a deployment descriptor `resource-env-ref` name.

Data type	String

Topic Space:

The name of the topic space. This field is required.

Data type	String
Default	DEFAULT_TOPIC_SPACE

Topic Name:

The name of the topic. This field is required.

Data type	String

Delivery Mode:

The default mode for messages sent to this destination.

Data type	String
Range	Application, Persistent or NonPersistent
Default	Application

Time to Live:

The default length of time from its dispatch time that a message sent to this destination should be retained by the system, where **0** indicates that time to live value does not expire. Value from the producer is used if not completed.

Data type	Long
Units	Milliseconds

Priority:

The priority for messages sent to this destination. Value from producer is used if not completed.

Data type	Integer
Range	0 to 9 with **0** as the lowest priority and **9** as the highest priority

Read Ahead:

Used to control read-ahead optimization during message delivery.

Data type	String
Range	AsConnection, AlwaysOn and AlwaysOff
Default	AsConnection

Version 5 Default Provider queue connection factory settings for application clients

Use this panel to browse or change the configuration properties of the selected JMS queue connection factory for point-to-point messaging for use by WebSphere Application Server version 5 applications. These configuration properties control how connections are created between the JMS provider and the default messaging system that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Provider** > **Version 5 Default Provider**. Right-click **Queue Connection Factories** and click **New**. The following fields appear on the **General** tab.

A queue connection factory is used to create JMS connections to queue destinations. The queue connection factory is created by the internal WebSphere Application Server product JMS provider. A Version 5 Default Provider queue connection factory has the following properties:

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

The User ID used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a User ID and password explicitly, for example, if the calling application uses the method createQueueConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Re-Enter Password:

Confirms the password.

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type String

Application Server:

Enter the name of the application server. This name is not the host name of the machine, but the name of the configured application server.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Version 5 Default Provider topic connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the internal product Java Message Service (JMS) provider. These configuration properties control how connections are created between the JMS provider and the messaging system that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Version 5 Default Provider**. Right click **Topic Connection Factories** and click **New**. The following fields appear on the **General** tab.

A Version 5 Default Provider topic connection factory has the following properties.

Name:

The name by which this queue connection factory is known for administrative purposes within WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere Application Server administrative domain.

Data type String

Description:

A description of this topic connection factory for administrative purposes within WebSphere Application Server.

Data type String

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly, for example, if the calling application uses the method createTopicConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Node:

The WebSphere Application Server node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type Enum
Range Pull-down list of nodes in the WebSphere Application Server administrative domain.

Application Server:

Enter the name of the application server. This name is not the host name of the machine, but the name of the configured application server.

Port:

Which of the two ports that connections use to connect to the JMS Server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for nonpersistent, nontransactional, nondurable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish or subscribe support. Therefore, any topic connection factory configured with the Port set to `Direct` cannot be used with message-driven beans.

Data type Enum
Default QUEUED

Range

QUEUED
The listener port used for full-function JMS compliant, publish or subscribe support.

DIRECT
The listener port used for direct TCP/IP connection (nontransactional, nonpersistent, and nondurable subscriptions only) for publish or subscribe support.

The TCP/IP port numbers for these ports are defined on the product internal JMS server.

Client ID:

The JMS client identifier used for connections to the MQSeries queue manager.

Data type String

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Version 5 Default Provider queue destination settings for application clients

Use this panel to view or change the configuration properties of the selected queue destination for use with product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Version 5 Default Provider**. Right click **Queue Destinations** and click **New**. The following fields are displayed on the **General** tab.

A queue destination is used to configure the properties of a JMS queue. A Version 5 Default Provider queue destination has the following properties.

Name:

The name by which the queue is known for administrative purposes within WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes.

Data type String

JNDI Name:

The application client run time uses this field to retrieve configuration information.

Persistence:

Whether all messages sent to the destination are persistent, nonpersistent, or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Nonpersistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest).

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or whether messages on the queue expire (have an unlimited expiry timeout).

Data type Enum

Default APPLICATION_DEFINED

Range

Application defined
The expiry timeout for messages in this queue is defined by the application that put them onto the queue.

Specified
The expiry timeout for messages in this queue is defined by the **Specified expiry** property.If you select this option, you must define a time out on the **Specified expiry** property.

Unlimited
Messages in this queue have no expiry timeout, and those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, specify the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type Integer

Units Milliseconds

Range Greater than or equal to 0

- 0 indicates that messages never timeout.
- Other values are an integer number of milliseconds.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Version 5 Default Provider topic destination settings for application clients

Use this panel to view or change the configuration properties of the selected topic destination for use with the internal product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Version 5 Default Provider**. Right click **Topic Destinations** and click **New**. The following fields appear on the **General** tab.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. A Version 5 Default Provider topic has the following properties.

Name:

The name by which the topic is known for administrative purposes.

Data type String

Description:

A description of the topic, for administrative purposes within WebSphere Application Server.

Data type String

JNDI Name:

The application client run-time environment uses this field to retrieve configuration information.

Topic Name: The name of the topic as defined to the JMS provider.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, nonpersistent, or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Nonpersistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to `Specified`, specify the message priority for this queue, in the range 0 (lowest) through 9 (highest).

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout).

Data type	Enum
Default	APPLICATION_DEFINED
Range	**Application defined** The expiry timeout for messages on this queue is defined by the application that put them onto the queue. **Specified** The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.* **Unlimited** Messages on this queue have no expiry timeout, so those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type	Integer
Units	Milliseconds
Range	Greater than or equal to 0 • 0 indicates that messages never time out. • Other values are an integer number of milliseconds.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere MQ Provider queue connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the MQSeries product Java Message Service (JMS) provider. These configuration properties control how connections are created between the JMS provider and WebSphere MQ.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right click **Queue Connection Factories**, and click **New**. The following fields are displayed on the **General** tab.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book, located in the WebSphere MQ Family library.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

A queue connection factory for the JMS provider has the following properties.

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly; for example, if the calling application uses the method createQueueConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type	String
Default	Null

Re-Enter Password:

Confirms the password.

Queue Manager:

The name of the MQSeries queue manager for this connection factory.

Connections created by this factory connect to that queue manager.

Data type	String

Host:

The name of the host on which the WebSphere MQ queue manager runs for client connection only.

Data type	String
Default	Null
Range	A valid TCP/IP host name

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type	Integer
Default	Null
Range	A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type	String
Default	Null
Range	1 through 20 ASCII characters

Transport type:

Specifies whether the WebSphere MQ client connection or JNDI bindings are used for connection to the WebSphere MQ queue manager. The external JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF nonpersistent, nondurable, nontransactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type Enum

Units Not applicable

Default BINDINGS

Range

BINDINGS
JNDI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and poses security risks that should be addressed through the use of EJB roles.

CLIENT
WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.

DIRECT
For WebSphere MQ Event Broker using DIRECT mode. DIRECT is a lightweight sockets protocol used in nontransactional, nondurable and nonpersistent Publish/Subscribe messaging. DIRECT only works for clients and message-driven beans using the non-ASF protocol.

QUEUED
QUEUED is a standard TCP protocol.

Recommended

Queue connection factory transport type
BINDINGS is faster by 30% or more, but it lacks security. When you have security concerns, BINDINGS is more desirable than CLIENT.

Topic connection factory transport type
DIRECT is the fastest type and should be used where possible. Use BINDINGS when you want to satisfy additional security tasks and the queue manager is local to the JMS client. QUEUED is the fallback for all other cases. WebSphere MQ 5.3 before CSD2 with the DIRECT setting can lose messages when used with message-driven beans and under load. This loss also happens with client-side applications unless the broker maxClientQueueSize is set to 0. You can set this to 0 with the command:

```
#wempschangeproperties WAS_nodeName_server1
 -e default -o DynamicSubscriptionEngine -n
 maxClientQueueSize -v 0 -x executionGroupUUID
```

where executionGroupUUID can be found by starting the broker and looking in the Event Log/Applications for event 2201. This value is usually ffffffff-0000-0000-000000000000.

Note: The WebSphere MQ 5.3 JMS cannot be used within WAS 6.1 because WAS 6.1 has a Java 5 runtime. Therefore, cross-memory connections cannot be established with WebSphere MQ 5.3 queue managers. This can result in a performance degradation if you were previously using WebSphere MQ 5.3 and BINDINGS for your connections and move to CLIENT network connections in migrating to WAS 6.1.

Client ID:

The JMS client identifier used for connections to the MQSeries queue manager.

Data type String

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These references are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Message Retention:

Select this check box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are handled according to their disposition options.

Data type Enum
Units Not applicable
Default Cleared
Range
Selected
Unwanted messages are left on the queue.
Cleared
Unwanted messages are handled according to their disposition options.

Temporary model:

The name of the model definition used to create temporary connection factories if a connection factory does not already exist.

Data type String
Range 1 through 48 ASCII characters

Temporary queue prefix:

The prefix used for dynamic queue naming.

Data type String

Fail if quiesce:

Specifies whether applications return from a method call if the queue manager has entered a controlled failure.

Data type Check box
Default Selected

Local Server Address:

Specifies the local server address.

Data type String

Polling Interval:

Specifies the interval, in milliseconds, between scans of all receivers during asynchronous message delivery

Data type	Integer
Units	Milliseconds
Default	5000

Rescan interval:

Specifies the interval in milliseconds between which a topic is scanned to look for messages that have been added to a topic out of order.

This interval controls the scanning for messages that have been added to a topic out of order with respect to a WebSphere MQ browse cursor.

Data type	Integer
Units	Milliseconds
Default	5000

SSL cipher suite:

Specifies the cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider. The value must match the CipherSpec specified on the SVRCONN channel as the **Channel** property.

You must set this property, if you set the **SSL Peer Name** property.

SSL certificate store:

Specifies a list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. If you specify a value for this property, you must use WebSphere MQ JVM at Java 2 version 1.4.

The value is a space-delimited list of entries of the form:
`ldap://hostname:[port]`

A single slash (/) follows this value. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section “Working with Certificate Revocation Lists” in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL peer name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connection time.

If this property is not set, such certificate checking is performed.

The SSL peer name property is ignored if **SSL Cipher Suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:
`CN=QMGR.*, OU=IBM, OU=WEBSPHERE`

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the section "Distinguished Names" in the WebSphere MQ Security book.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Data type	Check box
Default	Selected

WebSphere MQ Provider topic connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the WebSphere MQ product Java Message Service (JMS) provider. These configuration properties control how connections are created between the JMS provider and WebSphere MQ.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right-click **Topic Connection Factories** and click **New**.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ product JMS resources. For more information about configuring WebSphere MQ product JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

MA0C broker: When creating a WebSphere Application Server v6 topic connection factory for the MA0C broker, you should consider the following attribute values:

BrokerControlQueue
: This value is fixed at SYSTEM.BROKER.CONTROL.QUEUE for the MA0C broker and is the queue the broker reads from.

BrokerVersion
: Set this value to BASIC for the MA0C broker.

ClientID
: Set this value to whatever you like for the MA0C broker (the value is string and is merely an identifier for your client application).

XA Enabled
Set this value to TRUE or FALSE for the MA0C broker (the setting you use is a performance enhancement flag - you will probably want to set this to 'true' most of the time).

BrokerMessage Selection
This value is fixed at CLIENT for the MA0C broker because the broker relies on client side message selection.

Direct Broker Authorization Type
This value is not required by the MA0C broker.

A topic connection factory for the WebSphere MQ product JMS provider has the following properties.

Name:

The name by which this topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS provider.

Data type	String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String

JNDI Name:

The Java Naming and Directory Interface (JNDI) name that is used to bind the topic connection factory into the application server name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type	String
Units	En_US ASCII characters
Range	1 through 45 ASCII characters

User ID:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User** property, you must also specify a value for the **Password** property.

The connection factory **User** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly, for example, if the calling application uses the method createTopicConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Queue Manager:

The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type String

Host:

The name of the host on which the WebSphere MQ queue manager runs for client connections only.

Data type String
Range A valid TCP/IP host name

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type Integer
Range A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for client connections to the WebSphere MQ queue manager for client connection only.

Data type String
Range 1 through 20 ASCII characters

Transport Type:

Whether WebSphere MQ client connection or JNDI bindings are used for connection to the WebSphere MQ queue manager.

Data type Enum
Default BINDINGS
Range
CLIENT
WebSphere MQ client connection is used to connect to the WebSphere MQ queue manager.
BINDINGS
JNDI bindings are used to connect to the WebSphere MQ queue manager.

Client ID:

The JMS client identifier used for connections to the WebSphere MQ queue manager.

Data type String

CCSID:

The coded character set identifier to be used with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String

Broker Control Queue:

The name of the broker control queue to which all command messages (except publications and requests to delete publications) are sent.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Queue Manager:

The name of the WebSphere MQ queue manager that provides the Publisher and Subscriber message broker.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Publish Queue:

The name of the broker input queue that receives all publication messages for the default stream.

The name of the broker's input queue (stream queue) that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Subscribe Queue:

The name of the broker queue from which nondurable subscription messages are retrieved.

The name of the broker queue from which nondurable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker CCSubQ:

The name of the broker queue from which nondurable subscription messages are retrieved for a ConnectionConsumer request. This property applies only for use of the Web container.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Version:

Specifies whether the message broker is provided by the WebSphere MQ MA0C SupportPac or newer versions of WebSphere family message broker products.

Data type Enum
Default Advanced
Range
Advanced The message broker is provided by newer versions of WebSphere family message broker products (MQ Integrator and MQ Publish and Subscribe).
Basic The message broker is provided by the WebSphere MQ MA0C SupportPac (WebSphere MQ - Publish and Subscribe).

Cleanup level:

Specifies the level of clean up provided by the publish or subscribe cleanup utility.

Data type Enum
Default SAFE
Range
ASPROP
NONE
STRONG

Cleanup interval:

Specifies the interval, in milliseconds, between background executions of the publish/subscribe cleanup utility.

Data type Integer

Units Milliseconds
Default 6000

Message selection:

Specifies where broker message selection is performed.

Data type Enum
Default BROKER
Range

BROKER
Message selection is done at the broker location.

Message CLIENT
Message selection is done at the client location.

Publish acknowledge interval:

The interval, in number of messages, between publish requests that require acknowledgement from the broker.

Data type Integer
Default 25

Sparse subscriptions:

Enables sparse subscriptions.

Data type Check box
Default Cleared

Status refresh interval:

The interval, in milliseconds, between transactions to refresh publish or subscribe status.

Data type Integer
Default 6000

Subscription store:

Specifies where WebSphere MQ stores data relating to active JMS subscriptions.

Data type Enum
Default MIGRATE
Range

MIGRATE

QUEUE

BROKER

Multicast:

Specifies whether this connection factory uses multicast transport.

Data type Enum
Default NOT USED
Range

NOT USED
This connection factory does not use multicast transport.

ENABLED
This connection factory always uses multicast transport.

ENABLED_IF_AVAILABLE
This connection factory uses multicast transport.

ENABLED_RELIABLE
This connection factory uses reliable multicast transport.

ENABLED_RELIABLE_IF_AVAILABLE
This connection factory uses reliable multicast transport if available.

Direct authentication:

Specifies whether to use direct broker authorization.

Data type Enum
Default NONE
Range

NONE Direct broker authorization is not used.

PASSWORD
Direct broker authorization is authenticated with a password.

CERTIFICATE
Direct broker authorization is authenticated with a certificates.

Proxy Host Name:

Specifies the host name of a proxy to be used for communication with WebSphere MQ.

Data type String

Proxy Port:

Specifies the port number of a proxy to be used for communication with WebSphere MQ.

Data type Integer
Default 0

Fail if quiesce:

Specifies whether applications return from a method call if the queue manager has entered a controlled failure.

Data type Check box
Default Selected

Local Server Address:

Specifies the local server address.

Data type	String

Polling Interval:

Specifies the interval, in milliseconds, between scans of all receivers during asynchronous message delivery.

Data type	Integer
Units	Milliseconds
Default	5000

Rescan interval:

Specifies the interval in milliseconds between which a topic is scanned to look for messages that have been added to a topic out of order.

This interval controls the scanning for messages that have been added to a topic out of order with respect to a WebSphere MQ browse cursor.

Data type	Integer
Units	Milliseconds
Default	5000

SSL cipher suite:

Specifies the cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider. The value must match the CipherSpec specified on the SVRCONN channel as the **Channel** property.

You must set this property, if you set the **SSL Peer Name** property.

SSL certificate store:

Specifies a list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. If you specify a value for this property, you must use WebSphere MQ JVM at Java 2 version 1.4.

The value is a space-delimited list of entries of the form:

`ldap://`*`hostname`*`:[`*`port`*`]`

A single slash (/) follows this value. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section “Working with Certificate Revocation Lists” in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL peer name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connection time.

If this property is not set, such certificate checking is performed.

The SSL peer name property is ignored if **SSL Cipher Suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:

```
CN=QMGR.*, OU=IBM, OU=WEBSPHERE
```

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the section "Distinguished Names" in the WebSphere MQ Security book.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Data type	Check box
Default	Selected

WebSphere MQ Provider queue destination settings for application clients

Use this panel to view or change the configuration properties of the selected queue destination for use with the WebSphere MQ product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right-click **Queue Destinations** and click **New**. The following fields are displayed on the **General** tab.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ product for JMS resources. For more information about configuring WebSphere MQ product for JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters.

A queue for use with the WebSphere MQ product JMS provider has the following properties.

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes.

Data type String

JNDI Name:

The application client run-time environment uses this field to retrieve configuration information.

Persistence:

Whether all messages sent to the destination are persistent, nonpersistent or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Nonpersistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to `Specified`, specify the message priority for this queue, in the range 0 (lowest) through 9 (highest).

Data type Integer
Units Message priority level
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout value for this queue is defined by the application or the by **Specified expiry** property or whether messages on the queue never expire (have an unlimited expiry time out).

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range

Application defined
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

Specified
The expiry timeout for messages on this queue is defined by the **Specified expiry** property. If you select this option, you must define a timeout on the **Specified expiry** property.

Unlimited
Messages on this queue have no expiry timeout and those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type Integer
Units Milliseconds
Range Greater than or equal to 0
- 0 indicates that messages never time out
- Other values are an integer number of milliseconds

Base Queue Name:

The name of the queue to which messages are sent, on the queue manager specified by the **Base queue manager name** property.

Data type String

Base Queue Manager Name:

The name of the WebSphere MQ queue manager to which messages are sent.

This queue manager provides the queue specified by the **Base queue name** property.

Data type String

Units En_US ASCII characters
Range A valid WebSphere MQ Queue Manager name, as 1 through 48 ASCII characters

CCSID:

The coded character set identifier to use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSID identifier supported by WebSphere MQ queue manager.

Data type String

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

Indicates that if native encoding is not enabled to select whether decimal encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

Indicates that if native encoding is not enabled to select the type of floating point encoding.

Data type Enum
Default IEEENORMAL
Range
IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Native encoding:

Indicates that the queue destination use native encoding (appropriate encoding values for the Java platform) when you select this check box.

Data type Enum
Default Cleared
Range
Cleared
Native encoding is not used, so specify the following properties for integer, decimal and floating point encoding.
Selected
Native encoding is used (to provide appropriate encoding values for the Java platform).

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Target client:

Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application.

Data type Enum
Default WebSphere MQ
Range
WebSphere MQ
The target is a traditional WebSphere MQ application that does not support JMS.
JMS The target application supports JMS.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere MQ Provider topic destination settings for application clients

Use this panel to view or change the configuration properties of the selected topic destination for use with the WebSphere MQ product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right click **Topic Destinations**, and click **New**. The following fields are displayed on the **General** tab.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ product JMS resources. For more information about configuring WebSphere MQ product JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. A topic for use with the WebSphere MQ product JMS provider has the following properties.

Name:

The name by which the topic is known for administrative purposes.

Data type String

Description:

A description of the topic for administrative purposes within IBM WebSphere Application Server.

JNDI Name:

The application client run time uses this field to retrieve configuration information.

Persistence:

Specifies whether all messages sent to the destination are persistent, nonpersistent, or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them in the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Nonpersistent
Messages on the destination are not persistent.

Priority:

Specifies whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them in the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property. If you select this option, you must define a priority for the **Specified priority** property.

Specified Priority:

If the **Priority** property is set to `Specified`, type the message priority for this queue, in the range 0 (lowest) through 9 (highest).

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or by the **Specified expiry** property or by messages on the queue never expire (have an unlimited expiry timeout).

Data type Enum
Default APPLICATION_DEFINED
Range

Application defined
The expiry timeout for messages on this queue is defined by the application that put them in the queue.

Specified
The expiry timeout for messages in this queue is defined by the **Specified expiry** property. If you select this option, you must define a timeout value for the **Specified expiry** property.

Unlimited
Messages on this queue have no expiry timeout, and these messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, type the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type Integer
Units Milliseconds
Range Greater than or equal to 0
- 0 indicates that messages never time out.
- Other values are an integer number of milliseconds.

Base Topic Name:

The name of the topic to which messages are sent.

Data type String

CCSID:

The coded character set identifier to use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSID identifiers that WebSphere MQ supports.

Data type String

Units Integer
Range 1 through 65535

Integer encoding:

Indicates whether integer encoding is normal or reversed when native encoding is not enabled.

Data type Enum
Default NORMAL
Range
NORMAL
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

Indicates the type of floating point encoding when native encoding is not enabled.

Data type Enum
Default IEEENORMAL
Range
IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S/390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Native encoding:

Indicates that the queue destination uses native encoding (appropriate encoding values for the Java platform) when you select this check box.

Data type Enum
Default Cleared

Range

Cleared
Native encoding is not used, so specify the previous properties for integer, decimal and floating point encoding.

Selected
Native encoding is used (to provide appropriate encoding values for the Java platform).

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

BrokerDurSubQueue:

The name of the broker queue from which durable subscription messages are retrieved.

The subscriber specifies the name of the queue when it registers a subscription.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

BrokerCCDurSubQueue:

The name of the broker queue from which durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Target Client:

Specifies whether the receiving application is JMS compliant or is a traditional WebSphere MQ application.

Data type Enum
Default WebSphere MQ
Range

WebSphere MQ
The target is a traditional WebSphere MQ application that does not support JMS.

JMS The target is a JMS compliant application.

Multicast:

Specifies whether this connection factory uses multicast transport.

Data type Enum
Default AS_CF

Range

AS_CF This connection factory uses multicast transport.

DISABLED
This connection factory does not use multicast transport.

NOT_RELIABLE
This connection factory always uses multicast transport.

RELIABLE
This connection factory uses multicast transport when the topic destination is not reliable.

ENABLED
This connection factory uses reliable multicast transport.

Generic JMS connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected Java Message Service (JMS) connection factory for use with the associated JMS provider. These configuration properties control how connections are created between the JMS provider and the messaging system that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > *new_JMS_Provider_instance*. Right-click **Connection Factories**, and click **New**. The following fields are displayed on the **General** tab.

A Java Message Service (JMS) connection factory creates connections to JMS destinations. The JMS connection factory is created by the associated JMS provider. A JMS connection factory for a generic JMS provider (other than the internal default messaging provider or WebSphere MQ as a JMS provider) has the following properties:

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the associated JMS provider.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

Indicates the user ID used with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a userid and password explicitly; for example, if the calling application uses the method createQueueConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used with the **User ID** property for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String
Default Null

Re-Enter Password:

Confirms the password entered in the **Password** field.

External JNDI Name:

The JNDI name that is used to bind the queue into the application server name space.

As a convention, use the fully qualified JNDI name, for example, `jms/`*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI API by the platform.

Data type String

Connection Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publication or subscription).

Select one of the following options:
Queue
A JMS queue destination for point-to-point messaging.
Topic A JMS topic destination for publish subscribe messaging.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Generic JMS destination settings for application clients

Use this panel to view or change the configuration properties of the selected JMS destination for use with the associated JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > *new JMS Provider instance*. Right-click **Destinations**, and click **New**. The following fields are displayed on the **General** tab.

A JMS destination is used to configure the properties of a JMS destination for the associated generic JMS provider. Connections to the JMS destination are created by the associated JMS connection factory. A JMS destination for use with a generic JMS provider (not the default messaging provider or WebSphere MQ as a JMS provider) has the following properties.

Name:

The name by which the queue is known for administrative purposes within WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes.

JNDI Name:

The JNDI name of the actual (physical) name of the JMS destination bound into JNDI.

External JNDI Name:

The JNDI name that is used to bind the queue into the application server name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Destination Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publishing or subscribing).

Select one of the following options:

Queue
A JMS queue destination for point-to-point messaging.

Topic A JMS topic destination for pub/sub messaging.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring JMS provider, JMS connection factory and JMS destination settings for application clients

You can configure JMS Provider, JMS Connection Factory and JMS Destination settings. This topic provides the required fields, special cases, and an example.

The purpose of this article is to help you to configure JMS Provider, JMS Connection Factory and JMS Destination settings.

- Required fields include:
  - JMS Provider Properties page: name, and at least one protocol provider
  - JMS Connection Factory Properties page: name, jndiName, destination type
  - JMS Destination Properties page: name, jndiName, destination type
- Special cases:
  - The destination type must be `QUEUE`, or `TOPIC`.
- Example:

```
<resources.jms:JMSProvider xmi:id="JMSProvider_3" name="genericJMSProvider:name"
description="genericJMSProvider:description"
externalInitialContextFactory="genericJMSProvider:contextFactoryClass"
externalProviderURL="genericJMSProvider:providerUrl">
<classpath>genericJMSProvider:classpath</classpath>
<factories xmi:type="resources.jms:GenericJMSDestination"
xmi:id="GenericJMSDestination_1" name="jmsDestination:name"
jndiName="jmsDestination:jndiName" description="jmsDestination:description"
externalJNDIName="jmsDestination:externalJndiName" type="QUEUE">
<propertySet xmi:id="J2EEResourcePropertySet_15">
<resourceProperties xmi:id="J2EEResourceProperty_17" name="jmsDestination:customName"
value="jmsDestination:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms:GenericJMSConnectionFactory"
xmi:id="GenericJMSConnectionFactory_1" name="jmsCF:name" jndiName="jmsCF:jndiName"
description="jmsCF:description" userID="jmsCF:user" password="{xor}NTIsHBllMT4yOg=="
externalJNDIName="jmsCF:externalJndiName" type="QUEUE">
<propertySet xmi:id="J2EEResourcePropertySet_16">
<resourceProperties xmi:id="J2EEResourceProperty_18" name="jmsCF:customName"
value="jmsCF:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_17">
<resourceProperties xmi:id="J2EEResourceProperty_19"
name="genericJMSProvider:customName" value="genericJMSProvider:customValue"/>
</propertySet>
</resources.jms:JMSProvider>
```

Configuring new JMS connection factories for application clients

Use this task to create a new Java Message Service (JMS) connection factory configuration for your application client.

1. Click the JMS provider for which you want to create a connection factory in the tree. Complete one of the following actions:
   - Configure a new JMS provider.
   - Click an existing JMS provider.
2. Expand the JMS provider to view its **Connection Factories** folder.
3. Click the connection factory folder, and complete one of the following actions:
   - Right-click the folder and select**New**.
   - Click **Edit > New** on the menu bar.
4. Configure the JMS connection factory properties in the displayed fields.

5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring new Java Message Service destinations for application clients

Use this task to create a new Java Message Service (JMS) destination configuration for your application client.

1. Click the JMS provider in the tree for which you want to create a destination. Complete one of the following actions:
   - Configure a new JMS provider.
   - Click an existing JMS provider.
2. Expand the JMS provider to view its **Destinations** folder.
3. Click the provider folder, and complete one of the following actions:
   - Right-click the folder and select **New**.
   - Click **Edit > New** on the menu bar.
4. Configure the JMS destination properties in the displayed fields.
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring new resource environment providers for application clients

You can create new resource environment provider configurations for your application client using the Application Client Resource Configuration Tool (ACRCT).

During this task, you create new resource environment provider configurations for your application client.

To configure a new resource environment provider, perform the following steps:

1. Start the Application Configuration Resource Tool and open the EAR file for which you want to configure the new Java Message Service (JMS) provider. The EAR file contents display in a tree view.
2. Select from the tree the JAR file in which you want to configure the new JMS provider.
3. Expand the JAR file to view its contents.
4. Click the **Resource Environment Providers** folder. Take one of the following actions:
   - Right-click the provider folder, and click **New**.
   - Click **Edit > New** on the menu bar.
5. Configure the JMS provider properties in the displayed fields.
6. Click **OK** when you finish.
7. Click **File > Save** on the menu bar to save your changes.

Resource environment provider settings for application clients

Use this page to specify resource environment entry properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected Java Archive (JAR) file. Right-click **Resource Environment Providers**, and click **New**. The following fields are displayed on the **General** tab:

Name:

Specifies the administrative name for the resource environment provider.

Description:

Specifies a description of the resource environment provider for your administrative records.

Class Path:

Specifies the path to the JAR file that contains the implementation classes for the resource environment provider.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new resource environment entries for application clients

You can create new resource environment entries for your client application using the Application Client Resource Configuration Tool (ACRCT).

During this task, you create new resource environment entries for your client application.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new resource environment entry. The EAR file contents are in the displayed tree view.
3. Click the desired resource environment provider, and complete the following action to configure new providers:
   - Configure a new resource environment provider.
4. Expand the resource environment provider to view the **Resource Environment Entries** folder.
5. Click the resource environment entries folder, and complete one of the following actions:
   - Right-click the folder and select **New**.
   - Click **Edit > New** on the menu bar.
6. Configure the resource environment entry properties in the displayed fields.
7. Click **OK**.
8. Click **File > Save** on the menu bar to save your changes.

Resource environment entry settings for application clients

Use this page to specify resource environment entry properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Environment Providers** > *resource environment instance*. Right-click **Resource Environment Entries**, and click **New**. The following fields appear on the **General** tab:

Name:

Specifies the administrative name for the resource environment entry.

Description:

Specifies a description of the URL for your administrative records.

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

Use this name to link to the binding information of the platform. The binding associates the resources defined in the deployment descriptor of the module to the actual (or physical) resources bound into JNDI by the platform.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Managing application clients

You can manage J2EE application clients using the Application Client Resource Configuration Tool (ACRCT).

Perform the following tasks after deploying application clients. This task only applies to J2EE application clients.

1. Update data source and data source provider configurations.
2. Update URLs and URL provider configurations.
3. Update mail session configurations.
4. Update JMS provider, connection factories, and destination configurations.
5. Update MQ JMS provider, MQ connection factories and MQ destination configurations.
6. Update Resource Environment Entry and Resource Environment Provider configurations.
7. (Optional) Remove application client resources.

Updating data source and data source provider configurations with the Application Client Resource Configuration Tool

You can update the configuration of an existing data source or data source provider using the Application Client Resource Configuration Tool (ACRCT).

During this task, you update the configuration of an existing data source or data source provider. Perform this task when your database configuration changes.

1. Start the Application Client Resource Configuration Tool (ACRCT), and open the Enterprise Archive (EAR) file containing the data source or data source provider. The EAR file contents display in a tree view.
2. Select Java Archive (JAR) file from the navigation tree containing the data source or data source provider to update.
3. Expand the JAR file to view its contents until you locate the particular data source or data source provider to update. Take one of the following actions:
   - Right-click the data source object and click **Properties**.
   - Click **Edit > Properties** on the menu bar.
4. Update the properties in the displayed fields. For detailed field help, see:
   - Data source provider properties
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Updating URLs and URL provider configurations for application clients

You can update URLs and URL provider configurations for application clients using the Application Client Resource Configuration Tool (ACRCT).

1. Start the tool and open the Enterprise Archive (EAR) file containing the URL or URL provider. The EAR file contents are displayed in a tree view.
2. Select from the tree the Java Archive (JAR) file containing the URL or URL provider to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular URL or URL provider to update. Take one of the following actions:
   a. Right-click the URL object and click **Properties**.
   b. Click **Edit** > **Properties** on the menu bar.
5. Update the properties in the displayed fields.
6. Click **OK** when you finish.
7. Click **File** > **Save** on the menu bar to save your changes.

Updating mail session configurations for application clients

You can update the configuration of an existing JavaMail session using the Application Client Resource Configuration Tool (ACRCT).

During this task, you update the configuration of an existing JavaMail session. You cannot update the name of the default JavaMail provider, and you cannot delete the default JavaMail provider from the navigation tree.

1. Start the tool and open the Enterprise Archive (EAR) file containing the JavaMail session. The EAR file contents are displayed in the navigation tree view.
2. Select the Java Archive (JAR) file containing the JavaMail session to update from the navigation tree.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular JavaMail session to update. Take one of the following actions:
   a. Right-click the object and click **Properties**
   b. Click **Edit** > **Properties** from the menu bar.
5. Update the properties in the displayed fields.
6. Click **OK** when you finish.
7. Select **File** > **Save** from the menu bar to save your changes.

Updating Java Message Service provider, connection factories, and destination configurations for application clients

You can update the configuration of an existing Java Message Service (JMS) provider, connection factory or destination using the Application Client Resource Configuration Tool (ACRCT).

During this task, you update the configuration of an existing Java Message Service (JMS) provider, connection factory or destination.

1. Start the tool and open the Enterprise Archive (EAR) file containing the Java Message Service (JMS) provider, connection factory, or destination. The EAR file contents display in a tree view.
2. Select the Java Archive (JAR) file containing the JMS provider, connection factory, or destination to update from the navigation tree.
3. Expand the JAR file to view its contents until you locate the particular JMS provider, connection factory, or destination to update. When you find it, do one of the following actions:
   - Right-click the provider, and click **Properties**.
   - Click **Edit > Properties** on the menu bar.
4. Update the properties in the displayed fields. For detailed field help, see:

- JMS provider properties
- WebSphere Application Server Queue connection factory properties
- WebSphere Application Server Topic connection factory properties
- WebSphere Application Server Queue destination properties
- WebSphere Application Server Topic destination properties

5. Click **OK**.
6. Click **File > Save** to save your changes.

Updating WebSphere MQ as a Java Message Service provider, and its JMS resource configurations, for application clients

You can update an existing configuration of WebSphere MQ as a Java Message Service (JMS) provider, and update the configuration of WebSphere MQ connection factories or WebSphere MQ destinations.

Use this task to update an existing configuration of WebSphere MQ as a Java Message Service (JMS) provider, and to update the configuration of WebSphere MQ connection factories or WebSphere MQ destinations.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the Enterprise Archive (EAR) file containing the WebSphere MQ JMS provider, WebSphere MQ connection factory, or WebSphere MQ destination. The EAR file contents are displayed in the navigation tree view.
3. Select the Java Archive (JAR) file containing the JMS provider, connection factory, or destination to update.
4. Expand the JAR file to view its contents until you locate the particular JMS provider, connection factory, or destination that you want to update. Complete one of the following actions:
   - Right-click the appropriate object and click **Properties**.
   - Click **Edit > Properties** on the menu bar.
5. Update the properties in the displayed fields. For detailed field help, see:
   - JMS provider properties
   - MQ Queue connection factory properties
   - MQ Topic connection factory properties
   - MQ Queue destination properties
   - MQ Topic destination properties
6. Click **OK**.
7. Click **File > Save** to save your changes.

Updating resource environment entry and resource environment provider configurations for application clients

You can update the configuration of an existing resource environment entry or resource environment provider using the Application Client Resource Configuration Tool (ACRCT).

During this task, you update the configuration of an existing resource environment entry or resource environment provider.

1. Start the tool and open the Enterprise Archive (EAR) file containing the resource environment entry or resource environment provider. The EAR file contents display in a navigation tree view.
2. Select from the tree the Java Archive (JAR) file containing the resource environment entry or resource environment provider to update.
3. Expand the JAR file to view its contents until you locate the resource environment entry or resource environment provider to update. Take one of the following actions:
   - Right-click the resource environment object, and click **Properties**.
   - Click **Edit > Properties** on the menu bar.
4. Update the properties in the displayed fields. For detailed field help, see:
   - Resource environment provider properties
   - Resource environment entry properties

5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Example: Configuring Resource Environment settings:

You can configure Resource Environment settings. This topic provides the required fields and an example.

The purpose of this topic is to help you configure Resource Environment settings.
- Required fields:
  - Resource Environment Provider page: **Name**
  - Resource Environment Entry page: **Name, JNDI Name**
- Example:

```
<resources.env:ResourceEnvironmentProvider xmi:id="ResourceEnvironmentProvider_1"
name="resourceEnvProvider:name" description="resourceEnvProvider:description">
<classpath>resourceEnvProvider:classpath</classpath>
<factories xmi:type="resources.env:ResourceEnvEntry" xmi:id="ResourceEnvEntry_1"
name="resourceEnvEntry:name" jndiName="resourceEnvEntry:jndiName"
description="resourceEnvEntry:description">
<propertySet xmi:id="J2EEResourcePropertySet_20">
<resourceProperties xmi:id="J2EEResourceProperty_22"
name="resourceEnvEntry:customName" value="resourceEnvEntry:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_21">
<resourceProperties xmi:id="J2EEResourceProperty_23"
name="resourceEnvProvider:customName" value="resourceEnvProvider:customValue"/>
</propertySet>
</resources.env:ResourceEnvironmentProvider>
```

Example: Configuring resource environment custom settings for application clients:

You can configure resource environment custom settings.

The purpose of this topic is to help you configure resource environment custom settings.
- The custom page applies to every resource type. You can specify as many custom names and values as you need.
- Example:

```
<propertySet xmi:id="J2EEResourcePropertySet_20">
<resourceProperties xmi:id="J2EEResourceProperty_22"
name="resourceEnvEntry:customName" value="resourceEnvEntry:customValue"/>
</propertySet>
```

Removing application client resources

You can remove J2EE application client resources using the Application Client Resource Configuration Tool (ACRCT).

The option to delete an item does not offer a confirmation dialog. As a safeguard, consider saving your work right before you begin this task. If you change your mind after removing an item, you can close the EAR file without saving your changes, canceling your deletion. Remember to close the EAR file immediately after the deletion, or you also lose any unsaved work that you performed since the deletion.

This task only applies to J2EE application clients.

1. Start the Application Client Resource Configuration Tool (ACRCT) and open the Enterprise Archive (EAR) file from which you want to remove an object. The EAR file contents display in the navigation tree view. If you already have an EAR file open and have made some changes, click **File** > **Save** to save your work before preceding to delete an object.
2. Locate the object that you want to remove in the tree.
3. Right-click the object, and click **Delete**.

4. Click **File** > **Save**.

Chapter 12. Web services

Implementing Web services applications

This topic introduces you to using Web services that are based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification. WebSphere Application Server supports Web services that are developed and implemented based on Web Services for J2EE. Use Web services when operating across a variety of platforms, including the J2EE 1.4 and non-J2EE platforms.

Decide if a Web service implementation benefits your business process.

Implementing Web services applications is an easy way to integrate application systems together within or outside your company's infrastructure that otherwise function as a standalone systems. For example, your customer information database is a standalone application, but you want your accounting application to be able to access the customer data. You can create a Web service for the customer database and then enable the accounting application as a Web service client. The accounting application can now access the customer information. By implementing a Web service, these two applications can share information in an efficient manner.

Because Web services are easily applied to existing applications and information technology assets, new solutions can be deployed quickly and recomposed to address new opportunities. As Web services become more popular, the pool of services grows, promoting development of more robust models of just-in-time application and business integration over the Internet.

You can use Web services applications with WebSphere Application Server by following the steps provided:

1. Plan to use Web services.
2. (Optional) Migrate existing Web services.

   If you have used Web services based on Apache SOAP and now want to develop and implement Web services based on the Web Services for J2EE specification, you need to migrate client applications developed with all versions of 4.0, and versions of 5.0 prior to 5.0.2.
3. Develop Web services.

   This topic is a good starting point in learning about how to develop a J2EE Web service.
4. Configure Web services deployment descriptors.

   You need to configure the deployment descriptors so that WebSphere Application Server can process the incoming Web services requests.
5. Assemble Web services.

   This topic presents what you need to assemble a Web service and in what order you should assemble the parts, for example an enterprise archive (EAR) file.
6. Deploy Web services.

   This topic presents the steps necessary to deploy the EAR file that has been configured and enabled for Web services.
7. Configure Web service client bindings. This topic explains how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you must configure the client bindings to access the downstream Web service.
8. Publish the WSDL file.

   After installing a Web services application, and optionally modifying the endpoint information, you might need Web Services Description Language (WSDL) files containing the updated endpoint information. This topic presents the steps necessary to publish the WSDL files so that this information is available.
9. Develop Web services clients.

This topic explains how to develop a Web services client based on the Web Services for J2EE specification.

10. Secure Web services.

    This topic presents the methods used to integrate message-level security into a WebSphere Application Server environment. If you are using V5.x, refer to Securing Web services for version 5.x applications based on WS-Security. If you are using V6.x, refer to Securing Web services for version 6 applications based on WS-Security

11. Monitoring the performance of Web services applications.

    This topic includes information to help you use the Performance Monitoring Infrastructure (PMI) to measure the time required to process Web services requests.

12. Troubleshoot Web services.

    You can use this topic to learn more about troubleshooting different processes used to develop, implement and use Web services, including command-line tools, Java compiling errors, client runtime errors and exceptions, serialization and deserialization errors, and authentication challenges and authorization failures with Web services security.

The following example illustrates how a business might use Web services.

The owner of a flower shop wants to start receiving orders from customers through the Web. This owner starts the process by finding wholesale flower suppliers, pricing the product, and completing contracts for future flower orders.

Using Web services, the flower shop owner can find wholesale flower suppliers.

The flower shop owner can request price lists from each of the suppliers by obtaining a WSDL file for each potential supplier. The WSDL can be downloaded from the supplier's Web page, received through e-mail, or retrieved from the supplier's UDDI registry entry.

The WSDL describes the procedure call. When using WebSphere Application Server, the procedure call is a Java API for XML-based remote procedure call (JAX-RPC), which retrieves price lists. The WSDL file also specifies the Universal Resource Locator (URL) where the request is sent.

The flower shop owner now has to compare the prices received back from each supplier, decide which suppliers to do business with, and make arrangements for future orders to fill. The flower shop can now sell merchandise through the Web by using Web services to communicate with suppliers for the best prices and complete the ordering processes. The merchandise price lists need publishing to the Web site and a mechanism is needed for customers to order flowers.

The Web services clients of the flower supplier are deployed on the flower shop server. When a customer makes a transaction to purchase flowers through the Web, the order is sent to the supplier through JAX-RPC. The supplier responds by sending a confirmation with the order number and shipping date. The suppliers maintain the inventory and the flower shop owner handles billing and customer order management.

Similarly, the flower shop catalog can be composed automatically from the catalogs of all the suppliers. If the supplier ships directly to the customer, the order tracking inquiries can pass directly to the supplier's order tracking system. The supplier can also use Web services to send invoices for orders and by the flower shop to pay the supplier's invoices. Processes that previously required forms to fill manually, and fax or mail, can now be done automatically, saving labor costs for both the flower shop and the supplier.

Using Web services is beneficial because a much larger inventory is made available to the flower shop. No merchandise maintenance overhead exists, but the flower shop can offer their customers products that they otherwise might not have. Selling flowers through the Web increases capital for the flower shop without overhead of another store or money invested into additional product.

For a more detailed scenario, see Web services scenario: Overview which tells the story of a fictional online garden supply retailer named Plants by WebSphere and how they incorporated the Web services concept.

Web services

Web services are self-contained, modular applications that you can describe, publish, locate, and invoke over a network.

WebSphere Application Server supports Web services that are developed and implemented based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

A typical Web services scenario is a business application requesting a service from another existing application. The request is processed through a given Web address using SOAP messages over a HTTP, Java Message Service (JMS) transport or invoked directly as Enterprise JavaBeans (EJB). The service receives the request, processes it, and returns a response. Examples of a simple Web service include weather reports or getting stock quotes. The method call is synchronous, that is, it waits until the result is available. Transaction Web services, supporting quotes, business-to-business (B2B) or business-to-client (B2C) operations include airline reservations and purchase orders.

Web services can include the actual service or the client that accesses the service.

Web services are Web applications that help you be more flexible in your business processes by integrating with applications that otherwise do not communicate. The inner-library loan program at your local library is a good example of the Web services concept and its evolution. The Web service concept existed even before the term; the concept became widely accepted with the creation of the Internet. Before the Internet was created, you visited your library, searched the collections and checked out your books. If you did not find the book that you wanted, the librarian did a search for you by computer or phone and located the book at a nearby library. The librarian ordered the book for you and you picked it up after it was delivered to your local library. By incorporating Web services applications, you can streamline your library visit.

Now, you can search the local library collection and other local libraries at the same time. When other libraries provide your library with a Web service to search their collection (the service might have been provided through UDDI), your results yield their resources. Another Web service application might enable you to check the book out and get it sent to your home. Using Web services applications saves time and provides a convenience for you, as well as freeing the librarian to do other business tasks.

Web services reflect the service-oriented architecture (SOA) approach to programming. This approach is based on the idea of building applications by discovering and implementing network-available services, or by invoking the available applications to accomplish a task. Web services deliver interoperability, for example, Web services applications provide components created in different programming languages to work together as if they were created using the same language. Web services rely on existing transport technologies, such as HTTP, and standard data encoding techniques, such as Extensible Markup Language (XML), for invoking the implementation.

The key components of Web services include:

- Web Services Description Language (WSDL)

  WSDL is the XML-based file that describes the Web service. The Web service request uses this file to bind to the service.
- SOAP

  SOAP is the XML-based protocol that the Web service request uses to invoke the service.

For a more detailed scenario, see Web services scenario: Overview, which tells the story of a fictional online garden supply retailer named Plants by WebSphere, and how this retailer incorporated the Web services concept.

Web Services for J2EE specification

The *Web services for Java 2 Platform, Enterprise Edition (J2EE)* specification defines the programming model and run-time architecture for implementing Web services based on the Java language. Another name for the Web Services for J2EE specification is the Java Specification Requirements (JSR) 109. The specification includes open standards for developing and implementing Web services.

The Web Services for J2EE specification focuses on Extensible Markup Language (XML) remote procedure call (RPC) and the Java language, including representing XML-based interface definitions in the Java language; Java language definitions in XML-based definition languages, such as SOAP, and assembling.

The J2EE technology can be integrated with Web services in a variety of ways. J2EE components, for example, JavaBeans and enterprise beans, can be exposed as Web services. These services can be accessed by clients written in Java code or by existing Web service clients that are not written in Java code. J2EE components can also act as Web service clients.

The Web Services for J2EE specification is the preferred platform for Web-based programming because it provides open standards allowing different types of languages, operating systems and software to communicate seamlessly through the Internet.

For a Java application to act as Web service client, a mapping between the Web Services Description Language (WSDL) file and the Java application must exist. The mapping is defined by the Java API for XML-based RPC (JAX-RPC) specification.

You can use a Java component to implement a Web service by specifying the component interface and binding information in the WSDL file and designing the application server infrastructure to accept the service request.

This entire process encompassed is based on the Web Services for J2EE specification.

The specification brings with it the `webservices.xml` deployment descriptor specifically for Web services. You are responsible for providing various elements to the deployment descriptor, including:

- Port name
- Port service implementation
- Port service endpoint interface
- Port WSDL definition
- Port QName
- JAX-RPC mapping
- Handlers (optional)
- Servlet mapping (optional)

The Enterprise JavaBeans (EJB) 2.1 specification also states that for a Web service developed from a session bean, the EJB deployment descriptor, `ejb-jar.xml`, must contain the service-endpoint element. The service-endpoint value must be the same as that stated in the webservices.xml deployment descriptor. To learn more about the EJB 2.1 specification see Enterprise beans: Resources for learning.

Review the API documentation for a complete list of API's. You can also review several articles about the development of Web services at Web services: Resources for learning.

JAX-RPC

The *Java API for XML-based RPC (JAX-RPC)* specification enables you to develop SOAP-based interoperable and portable Web services and Web service clients. JAX-RPC 1.1 provides core APIs for

developing and deploying Web services on a Java platform and is a required part of the J2EE 1.4 platform. The J2EE 1.4 platform allows you to develop portable Web services. Web services can also be developed and deployed on J2EE 1.3 containers.

WebSphere Application Server implements JAX-RPC 1.1 standards.

The JAX-RPC standard covers the programming model and bindings for using Web Services Description Language (WSDL) for Web services in the Java language. JAX-RPC simplifies development of Web services by shielding you from the underlying complexity of SOAP communication.

On the surface, JAX-RPC looks like another instantiation of remote method invocation (RMI). Essentially, JAX-RPC allows clients to access a Web service as if the Web service was a local object mapped into the client's address space even though the Web service provider is located in another part of the world. The JAX-RPC is done by using the XML-based protocol SOAP, which typically rides on top of HTTP.

JAX-RPC defines the mappings between the WSDL port types and the Java interfaces, as well as between Java language and Extensible Markup Language (XML) schema types.

A JAX-RPC Web service can be created from a JavaBean or a enterprise bean implementation. You can specify the remote procedures by defining remote methods in a Java interface. You only need to code one or more classes that implement the methods. The remaining classes and other artifacts are generated by the Web service vendor's tools. The following is an example of a Web service interface:

```
package com.ibm.mybank.ejb;
import java.rmi.RemoteException;
import com.ibm.mybank.exception.InsufficientFundsException;
/**
 * Remote interface for Enterprise Bean: Transfer
 */
public interface Transfer_SEI extends java.rmi.Remote {
   public void transferFunds(int fromAcctId, int toAcctId, float amount)
      throws java.rmi.RemoteException;

}
```

The interface definition in JAX-RPC must follow specific rules:

- The interface must extend `java.rmi.Remote` just like RMI.
- Methods must throw `java.rmi.RemoteException`.
- Method parameters cannot be remote references.
- Method parameter must be one of the parameters supported by the JAX-RPC specification. The following list are examples of method parameters that are supported. For a complete list of method parameters see the JAX-RPC specification.
  - Primitive types: `boolean`, `byte`, `double`, `float`, `short`, `int` and `long`
  - Object wrappers of primitive types: `java.lang.Boolean`, `java.lang.Byte`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short`
  - `java.lang.String`
  - `java.lang.BigDecimal`
  - `java.lang.BigInteger`
  - `java.lang.Calendar`
  - `java.lang.Date`
- Methods can take value objects which consist of a composite of the types previously listed, in addition to aggregate value objects.

A client creates a stub and invokes methods on it. The stub acts like a proxy for the Web service. From the client code perspective, it seems like a local method invocation. However, each method invocation gets marshaled to the remote server. Marshaling includes encoding the method invocation in XML as prescribed by the SOAP protocol.

The following are key classes and interfaces needed to write Web services and Web service clients:

- **Service interface**: A factory for stubs or dynamic invocation and proxy objects used to invoke methods
- **ServiceFactory class**: A factory for Services.
- **`loadService`**

  The `loadService` method is provided in WebSphere Application Server Version 6.0 to generate the service locator which is required by a JAX-RPC implementation. If you recall, in previous versions there was no specific way to acquire a generated service locator. For managed clients you used a JNDI method to get the service locator and for non-managed clients, you were required to instantiate IBM's specific service locator `ServiceLocator service=new ServiceLocator(...);` which does not offer portability. The `loadService` parameters include:
  - **`wsdlDocumentLocation:`** A URL for the WSDL document location for the service or null.
  - **`serviceName`**: A qualified name for the service
  - **`properties`**: A set of implementation-specific properties to help locate the generated service implementation class.
- **`isUserInRole`**

  The `isUserInRole` method returns a boolean indicating whether the authenticated user for the current method invocation on the endpoint instance is included in the specified logical role.
  - **`role`**: The role parameter is a String specifying the name of the role.
- **Service**
- **Call interface**: Used for dynamic invocation
- **Stub interface**: Base interface for stubs

If you are using a stub to access the Web service provider, most of the JAX-RPC API details are hidden from you. The client creates a ServiceFactory (`java.xml.rpc.ServiceFactory`). The client instantiates a Service (`java.xml.rpc.Service`) from the ServiceFactory. The service is a factory object that creates the port. The port is the remote service endpoint interface to the Web service. In the case of DII, the Service object is used to create Call objects, which you can configure to call methods on the Web service's port.

Review the API documentation for a complete list of API's. You can also review several articles about the development of Web services at Web services: Resources for learning.

SOAP

SOAP is a specification for the exchange of structured information in a decentralized, distributed environment. As such, it represents the main way of communication between the three key actors in a service oriented architecture (SOA): service provider, service requestor and service broker. The main goal of its design is to be simple and extensible. A SOAP message is used to request a Web service.

WebSphere Application Server follows the standards outlined in SOAP 1.1.

SOAP was submitted to the World Wide Web Consortium (W3C) as the basis of the Extensible Markup Language (XML) Protocol Working Group by several companies, including IBM and Lotus. This protocol consists of three parts:

- An *envelope* that defines a framework for describing message content and processing instructions.
- A set of *encoding rules* for expressing instances of application-defined data types.
- A *convention* for representing remote procedure calls and responses.

SOAP is a protocol-independent transport and can be used in combination with a variety of protocols. In Web services that are developed and implemented with WebSphere Application Server, SOAP is used in combination with HTTP, HTTP extension framework, and Java Message Service (JMS). SOAP is also operating-system independent and not tied to any programming language or component technology.

As long as the client can issue XML messages, it does not matter what technology is used to implement the client. Similarly, the service can be implemented in any language, as long as the service can process SOAP messages. Also, both server and client sides can reside on any suitable platform.

Review the API documentation for a complete list of API's. You can also review several articles about the development of Web services at Web services: Resources for learning.

SOAP with Attachments API for Java interface

SOAP with Attachments API for Java (SAAJ) interface is used for SOAP messaging that works behind the scenes in the Java API for XML-based RPC (JAX-RPC) implementation. You can map XML to Java types with standards supported by the JAX-RPC specification, but there are limited XML schema types, therefore you can use the SOAPElement interface to create a *custom data binder*.

Web services use XML messages to exchange messages. These messages conform to XML schema and when developing We services applications, there are limited XML APIs to work with, for example, Document Object Model (DOM). It is more efficient to manipulate the Java objects and have the serialization and deserialization completed during run time. To manipulate the XML schema types, you need to map the XML schema types to Java types with a *custom data binder*.

Web services uses SOAP messages to represent remote procedure calls between the client and the server. In normal JAX-RPC flows, the SOAP message is deserialized into a series of Java value-type business objects that represent the parameters and return values. In addition, JAX-RPC provides APIs that support applications and handlers to manipulate the SOAP message directly. Because there are a limited number of XML schema types that are supported by JAX-RPC, the specification provides the SAAJ data model as an extension to manipulate the message.

The primary interface in the SAAJ model is javax.xml.soap.SOAPElement, also referred to as SOAPElement. Using this model, applications can process an SAAJ model that uses pre-existing DOM code. It is also easier to convert pre-existing DOM objects to SAAJ objects.

Messages created using SAAJ follow SOAP standards. A SOAP message is represented in the SAAJ model as a javax.xml.soap.SOAPMessage object. The XML content of the message is represented by a javax.xml.soap.SOAPPart object. Each SOAP part has a SOAP envelope. This envelope is represented by the SAAJ javax.xml.SOAPEnvelope object. The SOAP specification defines various elements that reside in the SOAP envelope; SAAJ defines objects for the various elements in the SOAP envelope.

The SOAP message can also contain non-XML data that is called attachments. These attachments are represented by SAAJ AttachmentPart objects that are accessible from the SOAPMessage object.

A number of reasons exist as to why handlers and applications use the generic SOAPElement API instead of a tightly bound mapping:

- The Web service might be a conduit to another Web service. In this case, the SOAP message is only forwarded.
- The Web service might manipulate the message using a different data model, for example a Service Data Object (SDO). It is easier to convert the message from a SAAJ Document Object Model (DOM) to a different data model.
- A handler, for example, a digital signature validation handler, might want to manipulate the message generically.

You might need to go a step further to map your XML schema types, because the SOAPElement interface is not always the best alternative for legacy systems. In this case you might want to use a generic programming model, such as Service Data Object (SDO), which is more appropriate for data-centric applications.

The XML schema can be configured to include a custom data binding that pairs the SDO or data object with the Java object. For example, the run time renders an incoming SOAP message into a SOAPElement interface and passes it to the customer data binder for more processing. If the incoming message contains an SDO, the run time recognizes the data object code, queries its type mapping to locate a custom binder, and builds the SOAPElement interface that represents the SDO code. The SOAPElement is passed to the SDOCustomBinder.

See Custom data binders for more information about the process of developing applications with SOAPElement, SDO and custom binders.

Review the API documentation for a complete list of API's. You can also review several articles about the development of Web services at Web services: Resources for learning.

Web services SOAP/JMS protocol

The Web services engine supports the use of a Java Message Service (JMS)-compliant messaging transport as an alternative to HTTP for communicating SOAP messages between clients and servers.

You can use SOAP/JMS if you need to provide implementations for the client or server components, and need to make sure that the implementations are interoperable with the client and server components provided by the Web services engine in WebSphere Application Server.

Client responsibilities

The client component is responsible for sending SOAP request messages and receiving SOAP response messages while adhering to the following protocol constraints:

- The client must use either a JMS TextMessage, for example, javax.jms.TextMessage, or a BytesMessage, for example, javax.jms.BytesMessage, to transmit the SOAP request message to the server. If the request message contains attachments, a BytesMessage must be used. If the request message does not contain attachments, the client can use a TextMessage or a BytesMessage. The WebSphere product client implementation uses only a BytesMessage for the request message due to the potential need to transmit attachments.
- The client must set the following properties on the JMS request message before sending the message to the destination queue or topic:
  - **contentType** - This property is similar to the Content-Type header found in an HTTP message and is used to describe the content type of the message. A text-only SOAP message, for example, a message with no attachments, is written as follows:

    ```
    text/xml; charset="UTF-8"
    ```

    The **contentType** property in a SOAP request message that contains attachments must be set as follows:

    ```
    multipart/related; type="text/xml"; start="<...content-id of first part...>"
    ```

    This example represents a multi-part message, where the first part is of type ″text/xml″ that contains the SOAP message. The other parts of the multi-part message contain various attachments. The HTTP 1.1 specification contains more information about the Content-Type header.
  - **targetService** - This property must be set to the targetService property value that is found in the JMS-style endpoint location URL for the request. This value is used by the server component to determine the port component in the target when dispatching the request.
  - **endpointURL** - This property must be set to the JMS endpoint URL associated with the request.

  - **transportVersion** - This property indicates the version number of the protocol used by the client and server. Set the value to 1 (one).
- If the SOAP request message represents a two-way request, the client component must set the JMS message's **replyTo** property to specify the queue that is used for the reply message. The JMS message's `setJMSReplyTo` method is used for this.
- If the SOAP request message represents a one-way request, the client component must not set the JMS message's **replyTo** property.
- The client component must be prepared to handle a reply message that is a BytesMessage or a TextMessage, regardless of the type of JMS message used to transmit the SOAP request. The WebSphere product implementation of the server component responds with the same type of JMS message that is received from the client, unless the response contains attachments and a BytesMessage must be used.
- The client component can assume that the reply message **correlation ID** matches the original request **message ID**.

Server responsibilities

The server component is responsible for receiving the SOAP request messages and sending the SOAP response messages while adhering to the following protocol constraints:

- The server must be prepared to receive a TextMessage or a BytesMessage. If the request contains attachments, a ByteMessage must be used. The WebSphere product implementation of the server component responds in kind when sending the reply message back to the client, unless the response contains attachments and a BytesMessage is used.
- The server component must process the SOAP request properly to produce an appropriate SOAP reply message.
- The server component must send a reply message back to the client only if the JMS request message's **replyTo** property is set.
- The server component must set the following properties in the JMS reply message before sending the message to the replyTo queue:
  - **contentType** - See the description for this property in the client responsibilities section in this article.
  - The **correlation ID** of the JMS reply message should be set to the **message ID** of the original JMS request message. This is done by calling the JMS message's `setJMSCorrelationID` method.
  - **transportVersion** - This property indicates the version number of the protocol used by the client and server. Set the value to 1 (one).

Example: SOAP request without attachments

The following example displays the results from calling the JMS message's `toString` method for a request message without attachments:

```
JMSMessage class: jms_bytes
JMSType:          null
JMSDeliveryMode:  2
JMSExpiration:    0
JMSPriority:      4
JMSMessageID:     ID:d438eebf04cb124aa25c5821110a134f0000000000000001
JMSTimestamp:     1092110476167
JMSCorrelationID: null
JMSDestination:   topic://NewsGroupTopic?topicSpace=FvtTopicSpace
JMSReplyTo:       null
JMSRedelivered:   false
JMS_IBM_System_MessageID: 6B6765B36943A18C_11000001
transportVersion: 1
JMSXUserID:
targetService: NGConsumerJMS
JMSXAppID: Service Integration Bus
endpointURL: jms:/topic?destination=jms/NewsGroupTopic&connectionFactory;
```

```
  =jms/NewsGroupTCF&targetService;=NGConsumerJMS

contentType: text/xml; charset=utf-8
3c736f6170656e763a456e76656c6f706520786d6c6e733a736f6170656e763d22687474703a2f2f
736368656d61732e786d6c736f61702e6f72672f736f61702f656e76656c6f70652f2220786d6c6e
...
```

The following example displays the payload from the previous message example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<postMessage><ngName xsi:type="xsd:string">news.current.events</ngName>
<msg xsi:type="xsd:string">This is a sample news item.</msg>
</postMessage>
</soapenv:Body>
</soapenv:Envelope>
```

Example: SOAP request with attachments

The following example displays the results from calling the JMS message's `toString` method for a request message with attachments:

```
JMSMessage class: jms_bytes
JMSType:          null
JMSDeliveryMode:  1
JMSExpiration:    1092086312310
JMSPriority:      4
JMSMessageID:     ID:4bb64ed64e7d813d59ba5fec110a134f0000000000000001
JMSTimestamp:     1092086012310
JMSCorrelationID: null
JMSDestination:   queue://Logger_Q
JMSReplyTo:       queue://_Q_6B6765B36943A18C_00000385
JMSRedelivered:   false
JMS_IBM_System_MessageID: 6B6765B36943A18C_10000001
transportVersion: 1
JMSXUserID:
targetService: WSLoggerJMS
JMSXAppID: Service Integration Bus
endpointURL: jms:/queue?
destination=jms/Logger_Q&connectionFactory=jms/Logger_CF&targetService=WSLoggerJMS
contentType: multipart/related; type="text/xml";
start="<945414389.1092086011970.IBM.WEBSERVICES@myhost1>";
boundary="----=_Part_0_247953397.1092086011970"
0d0a2d2d2d2d2d2d3d5f506172745f305f323437393533333939372e313039323038363031313937330
0d0a436f6e74656e742d547970653a20746578742f786d6c3b20636861727365743d5554462d380d
...
```

The following displays the payload from the previous message example:

```
Content-Type: multipart/related; type="text/xml";

       start="<945414389.1092086011970.IBM.WEBSERVICES@myhost1>";

       boundary="----=_Part_0_247953397.1092086011970"



------=_Part_0_247953397.1092086011970
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: binary
Content-Id: <945414389.1092086011970.IBM.WEBSERVICES@myhost1>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
<p499:InternationalizationContext soapenv:mustUnderstand="0"
 xmlns:p499="http://www.ibm.com/webservices/InternationalizationContext">
<Locales>
 <Locale>
  <LanguageCode>en</LanguageCode>
  <CountryCode>US</CountryCode>
 </Locale>
</Locales>
<TimeZoneId>America/Chicago</TimeZoneId>
</p499:InternationalizationContext>
</soapenv:Header>
<soapenv:Body>
 <sendJpegImage/>
</soapenv:Body>
<soapenv:Envelope>
------=_Part_0_247953397.1092086011970
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <jpegImageRequest=81380956150.1092086011880.IBM.WEBSERVICES@myhost1>
<...contents of jpeg image file...>
```

SOAP response

The following example displays the results from calling the JMS message's `toString` method for a SOAP reply message:

```
JMSMessage class: jms_bytes
JMSType:          null
JMSDeliveryMode:  2
JMSExpiration:    0
JMSPriority:      4
JMSMessageID:     null
JMSTimestamp:     0
JMSCorrelationID: ID:cdddb857f078a266eb9a972f110a134f0000000000000001
JMSDestination:   null
JMSReplyTo:       null
JMSRedelivered:   false
contentType:
  multipart/related;
  type="text/xml";
  start="<961368106530.1092112854745.IBM.WEBSERVICES@yackerjr>";
  boundary="----=_Part_0_1655006754.1092112854745"
0d0a2d2d2d2d2d2d3d5f506172745f305f313635353030363735342e313039323131323835343734
350d0a436f6e74656e742d547970653a20746578742f786d6c3b20636861727365743d5554462d38
...
```

Review the API documentation for a complete list of API's. You can also review several articles about the development of Web services at Web services: Resources for learning.

Web Services-Interoperability Basic Profile

The *Web Services-Interoperability (WS-I) Basic Profile* is a set of non-proprietary Web services specifications that promote interoperability.

WebSphere Application Server conforms to the WS-I Basic Profile 1.1.

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

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

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

See how to modify your encoding from UTF-8 to UTF-16 if you need to change from UTF-8 to UTF-16.

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

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

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

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

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

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

RMI-IIOP using JAX-RPC

Java API for XML-based Remote Procedure Call (JAX-RPC) is the Java standard API for invoking Web services through remote procedure calls. A transport is used by a programming language to communicate over the Internet. You can use protocols with the transport such as SOAP and Remote Method Invocation (RMI). You can use Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) with JAX-RPC to support non-SOAP bindings.

Using RMI-IIOP with JAX-RPC, enables WebSphere Java clients to invoke enterprise beans using a WSDL file and the JAX-RPC programming model instead of using the standard J2EE programming model. When a enterprise JavaBeans implementation is used to invoke a Web service, multiprotocol JAX-RPC permits the Web service invocation path to be optimized for WebSphere Java clients. Learn more about this by reviewing Using enterprise bean bindings to invoke an EJB from a Web services client.

Benefits of using the RMI/IIOP protocol instead of a SOAP- based protocol are:

- XML processing is not required to send and receive messages; Java serialization is used instead.
- The client JAX-RPC call can participate in a user transaction, which is not the case when SOAP is used.

WS-I Attachments Profile

The *Web Services-Interoperability (WS-I) Attachments Profile* is a set of non-proprietary Web services specifications that promote interoperability. This profile compliments the WS-I Basic Profile 1.1 to add support for interoperable SOAP messages with attachments-based Web services.

WebSphere Application Server conforms to the WS-I Attachments Profile 1.0.

Attachments are typically used to send binary data, for example, data that is mapped in Java code to java.awt.Image and javax.activation.DataHandler. The raw data can be sent in the SOAP message, however, this approach is inefficient because an XML parser has to scan the data as it parses the message.

The WS-I Attachments Profile provides a solution to the limitations that are presented by Web Services Description Language (WSDL) 1.1. Because WSDL 1.1 attachments are not part of the XML schema type space, they can be message parts only. As message parts, the attachments cannot be arrays or properties of Java beans. The profile defines the wsi:swaRef XML schema type.Use the wsi:swaRef XML schema type to overcome the limitations of WSDL 1.1 attachments.

The wsi:swaRef type is an extension of the xsd:anyURI type, where its value contains the content-ID of the attachment.

Review the API documentation for a complete list of API's. You can also review several articles about the development of Web services at Web services: Resources for learning.

Web services: Resources for learning

This topic provides relevant supplemental information about Web services-related topics.

The following topics provide extended reference information about Web services:

- Web services overview
- Developing Web services:

  This topic includes information about developing Web services that based on the Java 2 Platform, Enterprise Edition (J2EE) and Java API for XML-based remote procedure call (JAX-RPC) specifications.
- Performance

  Review this topic for information about key Web sites that discuss performance best practices.
- Universal Description Discovery and Integration (UDDI)

This topic is an overview about UDDI and information about the UDDI Java API.
- The Web Services Invocation Framework (WSIF)

  This topic includes a look into the Apache Software Foundation and its maintenance of WSIF.
- Web Services-Interoperability (WS-I) Basic Profile

  This topic is an overview about the WS-I Basic Profile.
- SOAP

  This topic is an overview about SOAP, information about the SOAP syntax and processing rules.
- Security

  This topic provides a roadmap to security, the WS-Security specification, best practices, a profile of the OASIS Security Assertion Markup Language (SAML) and more.
- Samples

  This topic is the Samples Gallery for WebSphere Application Server and Samples Central for UDDI and WSIF.

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 an 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 services overview

- WebSphere Version 6 Web Services Handbook

  This IBM Redbook describes the new concept of Web services from various perspectives. It presents the major building blocks on which Web services rely. Well-defined standards and new concepts are presented and discussed.
- Web services (r)evolution, Part 1

  This article focuses on the benefits and challenges of building Web services applications. Web services might be an evolutionary step in designing distributed applications, however, the technology is not without problems. Outlined are the difficulties developers face in creating a truly workable distributed system of Web services. This article also outlines author Graham Glass's plan for building peer-to-peer Web applications.

Developing Web services

- Java Web Services: SOAP with Attachments API for Java (SAAJ)

  This document describes the SOAP with Attachments API for Java (SAAJ) and how this API provides a standard way to send XML documents over the Internet from the Java platform.
- JSR 109: Implementing Enterprise Web services

  This document describes the J2EE specification.
- JAX-RPC: Core Web services API in the Java platform

  This document reviews the JAX-RPC specification which enables Java technology developers to develop SOAP-based interoperable and portable Web services.
- A developer introduction to the JAX-RPC specification, Part 1: Learn the ins and outs of the JAX-RPC type-mapping system. The JAX-RPC specification is an important step forward in the quest for Web services interoperability. This DeveloperWorks WebSphere article explains the mapping between WSDL and XML types and Java types. It explains how the JAX-RPC standard defines this feature and some of the important points on designing an interoperable type system.
- A developer introduction to JAX-RPC, Part 2: Mine the JAX-RPC specification to improve Web service interoperability. This DeveloperWorks WebSphere article explains how you can achieve the next level of Web service interoperability using the JAX-RPC standard client and server side interface definitions and message processing model. It includes information on developing JAX-RPC handlers and handler chains.

- Getting Started with JAX-RPC. This article explains the basic JAX-RPC programming concepts. It describes the JAX-RPC client and server programming models and illustrates simple examples. The article is intended to give developers a good grasp of how to use the JAX-RPC specification to develop or use Web services.
- Web Services Description Language

  This article is a detailed overview of Web Services Description Language (WSDL), which includes programming specifications.

Performance

The following Web sites offer tips and best practices to get the best performance from your Web services applications:

- Best practices for Web services: Part 1, Back to the basics
- SOA and Web services: Articles
- IBM WebSphere Developer Technical Journal: Web Services Architectures and Best Practices
- Web services programming tips and tricks: How to create a simple JAX-RPC handler
- Web services programming tips and tricks: Using SOAP headers with JAX-RPC
- Web services programming tips and tricks: Extend JAX-RPC Web services using SOAP headers
- Web services programming tips and tricks: Roundtrip issues in Java coding conventions

UDDI

- Universal Description, Discovery and Integration

  This article is a detailed overview of the UDDI registry.
- A new approach to UDDI and WSDL: Introduction to the new OASIS UDDI WSDL Technical Note

  This article is about using WSDL with UDDI. Although it is based on the UDDI Registry in WebSphere Application Server Version 5, it remains a useful description of the recommended approach for use of WSDL with UDDI.
- UDDI Version 3 Features List

  This article is an introduction to the new features in UDDI Version 3.

WSIF

- The Apache Software Foundation. The Apache Software Foundation provides support for the Apache community of open-source software projects. Of particular interest is the Apache Web services project. The WSIF source code is donated by IBM to the Apache Software Foundation, and is maintained here as an Apache project.

WS-I Basic Profile

- Web Services Interoperability Organization This Web site offers resources and guidelines for Web services interoperability. You can also view the latest specification documents for WS-I Basic Profile from the documentation link on the home page.
- UTF and BOM Frequently Asked Questions. This Web site offers general information about UTF-8, UTF-16, UTF-32, along with Byte Order Mark (BOM) in a question and answer format.

SOAP

- SOAP

  This article is a detailed overview of SOAP, which includes the programming specifications.
- SOAP Security Extensions: Digital Signature

  This document specifies the syntax and processing rules of a SOAP header entry to carry digital signature information within a SOAP 1.1 Envelope

Security

- Security in a Web Services World: A Proposed Architecture and Roadmap

  This document describes a proposed model for addressing security within a Web service environment. It defines a comprehensive Web services security model that supports, integrates, and unifies several popular security models, mechanisms, and technologies, including both symmetric and public key technologies. You can enable a variety of systems to securely interoperate in a platform and language-neutral manner. It also describes a set of specifications and scenarios that show how these specifications can be used together.
- Web Services Security (WS-Security)

  The Web Services Security (WS-Security) specifications describe enhancements to SOAP messaging to provide the quality of protection through message integrity, message confidentiality, and single message authentication. Use these mechanisms to accommodate a wide variety of security models and encryption technologies. The WS-Security specification also provides a general-purpose mechanism for associating security tokens with messages. Additionally, the specification describes how to encode binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets, as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the credentials that are included with a message.
- SOAP Security Extensions: Digital Signature

  This document specifies the syntax and processing rules of a SOAP header entry to carry digital signature information within a SOAP 1.1 envelope
- Web Services Security Addendum

  This document describes clarifications, enhancements, best practices, and errata of the WS-Security specification.
- WS-Security Profile of the OASIS Security Assertion Markup Language (SAML) Working Draft 04, 10 September 2002

  This document proposes a set of standards for SOAP extensions that are used to increase message confidentiality.
- Web Services Security: SOAP Message Security Working Draft 12, Monday 21 April 2003

  This document describes the support for multiple token formats, trust domains, signature formats, and encyrption technologies.
- JSR 55:Certification Path API

  This document provides a short description of the Certification Path API.
- XML-Signature Syntax and Processing

  This document specifies XML digital signature processing rules and syntax. XML signatures provide integrity, message authentication, or signer authentication services for data of any type, whether it is located within the XML that includes the signature or elsewhere.
- Canonical XML Version 1.0

  This specification describes a method for generating a physical representation or the canonical form of an XML document that accounts for the permissible changes.
- Exclusive XML Canonicalization Version 1.0

  Canonical XML [XML-C14N] specifies a standard serialization of XML that, when applied to a subdocument, includes the subdocument ancestor context including all of the namespace declarations and attributes in the ″xml:″namespace.
- XML Encryption Syntax and Processing

  This document specifies a process for encrypting data and representing the result in XML.
- Decryption Transform for XML Signature

  This document specifies an XML Signature decryption transform that enables XML Signature applications to distinguish between those XML encryption structures that are encrypted before signing, and must not be decrypted, and those that are encrypted after signing, and must be decrypted, for the signature to validate.
- WS-Security

This document specifies resources for the April 2002 WS-Security specification. The following addenda and drafts are available:
  - http://schemas.xmlsoap.org/ws/2002/07/secext/
  - http://schemas.xmlsoap.org/ws/2002/07/utility/
  - OASIS draft 12 for secext
  - OASIS draft 12 for utility
- Specification: Web Services Security (WS-Security) Version 1.0 05 April 2002
- XML Encryption Syntax and Processing W3C Recommendation 10 December 2002
- XML-Signature Syntax and Processing W3C Recommendation 12 February 2002
- Web Services Security Addendum
- Web Services Security Core Specification Working Draft 01, 20 September 2002
- Web Services Security: SOAP Message Security Working Draft 13, Thursday, 01 May 2003
- Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC3280, April 2002
- OASIS Web Services Security Technical Committee

Samples
- Samples Gallery
- Samples Central. Samples and associated documentation for the following Web services components are available through the Samples Central page of the DeveloperWorks WebSphere Web site:
  - The IBM WebSphere UDDI Registry.
  - The Web Services Invocation Framework (WSIF).

Deploying Web services

This task explains how to deploy a Web service into WebSphere Application Server.

To deploy Web services that are based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification, you need an enterprise application, also known as an enterprise archive (EAR) file that is configured and enabled for Web services.

If you have a Web service that was deployed on a previous version of WebSphere Application Server, you might want to run the **wsdeploy** command-line tool so that you can benefit from performance features that have been added to this release.

This task is one of the steps in developing and implementing Web services.

You can use either the administrative console or the **wsadmin** scripting tool to deploy an EAR file. If you are installing an application containing Web services by using the **wsadmin** command, specify the -deployws option. If you are installing an application containing Web services by using the administrative console, select **Deploy WebServices** in the Install New Application wizard. For more information about installing applications using the administrative console see Installing a new application.

If the Web services application is previously deployed with the **wsdeploy** command, it is not necessary to specify Web services deployment during installation.

The following actions deploy the EAR file with the **wsadmin** command:

1. Start *install_root*/bin/wsadmin from a command prompt. If you are using Linux or Unix platforms, start *install_root*/bin/wsadmin.sh.
2. Enter the **$AdminApp install *EARfile* "-usedefaultbindings -deployws"** command at the **wsadmin** prompt.

You have a Web service installed into the WebSphere Application Server product.

You can confirm that the Web services application was deployed by entering the Web service endpoint URL in a browser, then viewing an informative page. The information page contains the following information:

```
{http://webservice.pli.tc.wssvt.ibm.com}RetireWebServices
Hi there, this is a Web service!
```

The first line of this information is variable, depending on your Web service. The URI in the brackets is the namespace and the string following that (in this example, RetireWebServices), is the name of the port used to access the Web service.

The next step you might want to consider is to apply security to the applications.

Provide options to perform the Web services deployment settings

Use this panel to specify options for Web services deployment.

This administrative console panel is a step in the application installation and update wizards. To view this panel, you must select **Deploy Web services** on the **Select installation options** panel. Thus, to view this panel, click **Applications > Install New Application >** *application_path* **> Show me all installation options and parameters > Next > Next > Deploy Web services > Next > Step: Provide options to perform the Web services deployment**.

You can specify the Web services deployment options on this panel only when installing or updating an application that uses Web services.

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 executed during installation after you click **Finish** on the **Summary** panel of the wizard.

Related tasks

“Installing application files with the console” on page 32
Installing 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*.

Related reference

“wsdeploy command” on page 385

“Enterprise application settings” on page 55
Use this page to configure an enterprise application.

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`*.

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`*.

Data type	String
Default	null

wsdeploy command

This topic explains how to use the **wsdeploy** command-line tool with Web services that are based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification. The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file. These classes include:

- Stubs
- Serializers and deserializers
- Implementations of service interfaces

This deployment step must be performed at least once, and can be performed more often. Deployment can be performed separately using the **wsdeploy** command, assembly tools, or when the application is installed. When using the **wsadmin** command for installation, specify the **-deployws** option.

The **wsdeploy** command operates as noted in the following list:

- Each module in the enterprise application or JAR file is examined.
- If the module contains Web services implementations, indicated by the presence of the `webservices.xml` deployment descriptor, the associated Web Services Description Language (WSDL) files are located and the **WSDL2Java** command is run with the role deploy-server option.
- If the module contains Web services clients, indicated by the presence of the client deployment descriptor, the associated WSDL files are located and the **WSDL2Java** command is run with the role deploy-client option.
- The files generated by the **WSDL2Java** command are compiled and repackaged.

See **WSDL2Java** command for more information about the files that are generated for deployment.

When the generated files are compiled, they can reference application-specific classes outside the EAR or JAR file, if the EAR or JAR file is not self-contained. In this case, use either the -jardir or -cp option to specify additional JAR or zip files to be added to CLASSPATH variable when the generated files are compiled.

wsdeploy command syntax

The command syntax is noted in the following example:

`wsdeploy` *`Input_filename Output_filename`* `[options]`

Required options:

- ***Input_filename***

  Specifies the path to the EAR or JAR file to deploy.
- ***Output_filename***

  Specifies the path of the deployed EAR or JAR file. If *output_filename* already exists, it is silently overwritten. The *output_filename* can be the same as the *input_filename*.

Other options:

- **-jardir** *directory*

  Specifies a directory that contains JAR or zip files. All JAR and zip files in this directory are added to the CLASSPATH used to compile the generated files. This option can be specified zero or more times.
- -**cp** *entries*

  Specifies entries to add to the CLASSPATH when the generated classes are compiled. Multiple entries are separated the same as they are in the CLASSPATH environment variable, with a semicolon on Windows platforms and a colon for Linux and Unix platforms.
- **-codegen**

  Specifies to generate but not compile deployment code. This option implicitly specifies the -keep option.
- **-debug**

  Includes debugging information when compiling, that is, use javac -g to compile.
- **-help**

  Displays a help message and exit.
- **-ignoreerrors**

  Do not stop deployment if validation or compilation errors are encountered.
- **-keep**

  Do not delete working directories containing generated classes. A message is displayed indicating the name of the working directory that is retained.
- **-novalidate**

  Do not validate the Web services deployment descriptors in the input file.
- **-trace**

  Displays processing information, including the names of the generated files.

Example The following example illustrates how the options are used with the **wsdeploy** command:

```
wsdeploy x.ear x_deployed.ear -trace -keep
Processing web service module x_client.jar.
Keeping directory: f:\temp\Base53383.tmp for module: x_client.jar.
Parsing XML file:f:\temp\Base53383.tmp\WarDeploy.wsdl
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java.
Done processing module x_client.jar.
```

Messages

- Flag *-f* is not valid.

  Option *f* was not recognized as a valid option.
- Flag *-c* is ambiguous.

  Options can be abbreviated, but the abbreviation must be unique. In this case, the **wsdeploy** command cannot determine which option was intended.
- Flag *-c* is missing parameter *-p*.

  A required parameter for an option is omitted.
- Missing *p* parameter.

  A required option is omitted.

Configuring Web service client bindings

When a Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise JavaBean (EJB) module, including client bindings.

Deploy the Web service into WebSphere Application Server.

To complete this task, you need to know the topology of the URL endpoint address of the Web services servers and which Web service the client depends upon. You can view the deployment descriptors in the administrative console to find topology information. See the article View Web services server deployment descriptors for more information.

The client bindings define the Web Services Description Language (WSDL) file name and preferred ports. The relative path of a Web service in a module is specified within a compatible WSDL file that contains the actual URL to be used for requests. The address is only needed if the original WSDL file did not contain a URL, or when a different address is needed. For a service endpoint with multiple ports, you need to define an alternative WSDL file name.

The following steps describe how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you must configure the client bindings to access the downstream Web service.

You can also configure client bindings with **wsadmin**.

To configure client bindings through the administrative console:

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* > **Web services client bindings**.
3. Find the Web service you want to update.

   The Web services are listed in the **Web Service** field.
4. Select the WSDL file name from the drop down box in the WSDL file name field.
5. Click **Edit** in the Preferred port mappings field to configure the default port to use.
   a. Specify the port type and the preferred ports in the Port type and Preferred ports fields.

      Configuring the preferred port enables you to select an optimal port implementation use non-SOAP protocols. See RMI-IIOP Web services using JAX-RPC for more information about using non-SOAP protocols.
   b. Click **Apply** and **OK**.
6. Click **Edit** in the Port information field to configure the request timeout, the overridden endpoint, and the overridden binding namespace for a port.

   Configuring the request timeout accommodates complex topologies that can have multiple cascaded Web services that involve multiple hops or long-running services.

   Timeout values can be configured based on observed behavior of the overall system as integration proceeds. For example, a Web service client might time out because of changing network conditions or the performance of an external Web service. When you have applications containing Web services clients that timeout, you can change the request time out values for the clients.
   a. Click **Apply** and **OK**.

Your Web service client bindings are configured.

Now you can finish any other configurations, start or restart the application, and verify the expected behavior of the Web service.

Web services client bindings

The client bindings define the Web Service Description Language (WSDL) file name, preferred ports and other port information. Use this page to specify the client bindings and the port mappings for the Web services in a module.

A Web service can specify the relative path within the module of a compatible WSDL file containing the actual URL to be used for requests. The relative path only needs to be specified if the original WSDL file

does not contain a URL or when a different URL is needed. For a service endpoint with multiple ports defined, a preferred mapping specifies the default port to use for a port type.

Web service
Identifies the name of this Web service. A module can contain one or more Web services.

EJB
Identifies the name of the EJB for the EJB modules.

WSDL file name
Specifies the WSDL file name, which is relative to the module. Locate the WSDL file name in the drop down menu.

Preferred port mappings
Specifies and manages the preferred port type mapping for a Web service when a particular port type is requested.

Click **Edit** to edit the preferred port mapping information on the **Preferred port mappings** panel.

Port information
Specifies additional configuration information for the ports of this Web service.

Click **Edit** to edit the port information on the **Port information** panel. You can set a request timeout, override an endpoint and override a binding namespace for each client port.

Preferred port mappings
Use this page to view and manage a preferred portType mapping for a Web service.

When you have multiple ports that reference the same portType (service endpoint interface), a preferred port specifies the port to use when the `Service.getPort(Class SEI)` method is called with only the service endpoint interface.

To view this administrative console page, click **Applications** >**Enterprise Application** > *application_instance* > **Manage Modules** > *module_instance* >**Web services client bindings** > **Edit** > *preferred_port_instance*.

portType:

Specifies the portType.

The preferred port and the portType values are both of the type `java.xml.namespace.QName`.

Preferred port:

Specifies the preferred port to be associated with a particular portType. The `Service.getPort(Class)` method returns the preferred port associated with the specified service endpoint interface class (portType).

The preferred ports available are listed, as well as a value of `None`, which indicates no preferred port is selected.

Web services client port information
Use this page to specify a request timeout, override an endpoint, and override a binding namespace for a Web services client port.

A Web service can have multiple ports. You can view and configure the port attributes for each defined Web service port. The Web services are listed on the Web services client bindings panel.

To view this page, click **Applications** >**Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* >**Web services client bindings** > **Edit**.

For EJB modules, click **Applications** >**Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* >**Web services client bindings** > **Edit**.

Port:

Specifies the name of a port.

Request timeout:

Specifies the time, in seconds, that a Web service client waits for a request to complete on this port. If a timeout is not specified, the default request timeout for the client to wait is `360` seconds. If the value is set at `0` (zero), the client's request does not timeout.

A typical use for this setting is to customize the client's behavior when it is configured to use a JMS transport to access a Web service to make it wait longer for an expected completion. Depending upon network conditions, or the nature of a Web service implementation, it might be necessary to tune the timeout.

Overridden endpoint:

Specifies the name of an endpoint that is used to override the current endpoint. A client invoking a request on this port uses this endpoint instead of the endpoint specified in the WSDL file.

If an assembled application contains a Web service client that is statically bound, the client is locked into using the implementation (service end point) identified in the WSDL file used during development. Overriding the endpoint is an alternative to configuring the deployed WSDL attribute.

The overridden endpoint URI attribute is specified on a per port basis. It does not require an alternative WSDL file within the module. The overridden endpoint URI takes precedence over the deployed WSDL attribute. The client uses this value for the service end point URI or SOAP address, instead of the value in the static client bindings.

Overridden binding:

Specifies the WSDL file binding namespace URI to use with this port, instead of the namespace in the WSDL file. This binding does not need to exist in the WSDL file. A client invoking a request on this port uses this binding instead of the binding specified in the WSDL file. An overridden binding namespace cannot be specified unless an overridden endpoint is specified.

Deploying Web services

This task explains how to deploy a Web service into WebSphere Application Server.

To deploy Web services that are based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification, you need an enterprise application, also known as an enterprise archive (EAR) file that is configured and enabled for Web services.

If you have a Web service that was deployed on a previous version of WebSphere Application Server, you might want to run the **wsdeploy** command-line tool so that you can benefit from performance features that have been added to this release.

This task is one of the steps in developing and implementing Web services.

You can use either the administrative console or the **wsadmin** scripting tool to deploy an EAR file. If you are installing an application containing Web services by using the **wsadmin** command, specify the -deployws option. If you are installing an application containing Web services by using the administrative console, select **Deploy WebServices** in the Install New Application wizard. For more information about installing applications using the administrative console see Installing a new application.

If the Web services application is previously deployed with the **wsdeploy** command, it is not necessary to specify Web services deployment during installation.

The following actions deploy the EAR file with the **wsadmin** command:

1. Start *install_root*/bin/wsadmin from a command prompt. If you are using Linux or Unix platforms, start *install_root*/bin/wsadmin.sh.
2. Enter the **$AdminApp install *EARfile* "-usedefaultbindings -deployws"** command at the **wsadmin** prompt.

You have a Web service installed into the WebSphere Application Server product.

You can confirm that the Web services application was deployed by entering the Web service endpoint URL in a browser, then viewing an informative page. The information page contains the following information:

```
{http://webservice.pli.tc.wssvt.ibm.com}RetireWebServices
Hi there, this is a Web service!
```

The first line of this information is variable, depending on your Web service. The URI in the brackets is the namespace and the string following that (in this example, RetireWebServices), is the name of the port used to access the Web service.

The next step you might want to consider is to apply security to the applications.

Provide options to perform the Web services deployment settings

Use this panel to specify options for Web services deployment.

This administrative console panel is a step in the application installation and update wizards. To view this panel, you must select **Deploy Web services** on the **Select installation options** panel. Thus, to view this panel, click **Applications > Install New Application >** *application_path* **> Show me all installation options and parameters > Next > Next > Deploy Web services > Next > Step: Provide options to perform the Web services deployment**.

You can specify the Web services deployment options on this panel only when installing or updating an application that uses Web services.

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 executed during installation after you click **Finish** on the **Summary** panel of the wizard.

Related tasks

"Installing application files with the console" on page 32
Installing 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*.

Related reference

"wsdeploy command" on page 385

"Enterprise application settings" on page 55
Use this page to configure an enterprise application.

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`*.

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`*.

Data type	String
Default	null

wsdeploy command

This topic explains how to use the **wsdeploy** command-line tool with Web services that are based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification. The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file. These classes include:

- Stubs
- Serializers and deserializers
- Implementations of service interfaces

This deployment step must be performed at least once, and can be performed more often. Deployment can be performed separately using the **wsdeploy** command, assembly tools, or when the application is installed. When using the **wsadmin** command for installation, specify the **-deployws** option.

The **wsdeploy** command operates as noted in the following list:

- Each module in the enterprise application or JAR file is examined.
- If the module contains Web services implementations, indicated by the presence of the `webservices.xml` deployment descriptor, the associated Web Services Description Language (WSDL) files are located and the **WSDL2Java** command is run with the role deploy-server option.
- If the module contains Web services clients, indicated by the presence of the client deployment descriptor, the associated WSDL files are located and the **WSDL2Java** command is run with the role deploy-client option.
- The files generated by the **WSDL2Java** command are compiled and repackaged.

See **WSDL2Java** command for more information about the files that are generated for deployment.

When the generated files are compiled, they can reference application-specific classes outside the EAR or JAR file, if the EAR or JAR file is not self-contained. In this case, use either the -jardir or -cp option to specify additional JAR or zip files to be added to CLASSPATH variable when the generated files are compiled.

wsdeploy command syntax

The command syntax is noted in the following example:

```
wsdeploy Input_filename Output_filename [options]
```

Required options:

- ***Input_filename***

  Specifies the path to the EAR or JAR file to deploy.
- ***Output_filename***

  Specifies the path of the deployed EAR or JAR file. If *output_filename* already exists, it is silently overwritten. The *output_filename* can be the same as the *input_filename*.

Other options:

- **-jardir** *directory*

  Specifies a directory that contains JAR or zip files. All JAR and zip files in this directory are added to the CLASSPATH used to compile the generated files. This option can be specified zero or more times.
- -**cp** *entries*

  Specifies entries to add to the CLASSPATH when the generated classes are compiled. Multiple entries are separated the same as they are in the CLASSPATH environment variable, with a semicolon on Windows platforms and a colon for Linux and Unix platforms.
- **-codegen**

  Specifies to generate but not compile deployment code. This option implicitly specifies the -keep option.
- **-debug**

  Includes debugging information when compiling, that is, use javac -g to compile.
- **-help**

  Displays a help message and exit.
- **-ignoreerrors**

  Do not stop deployment if validation or compilation errors are encountered.
- **-keep**

  Do not delete working directories containing generated classes. A message is displayed indicating the name of the working directory that is retained.
- **-novalidate**

  Do not validate the Web services deployment descriptors in the input file.
- **-trace**

  Displays processing information, including the names of the generated files.

Example The following example illustrates how the options are used with the **wsdeploy** command:

```
wsdeploy x.ear x_deployed.ear -trace -keep
Processing web service module x_client.jar.
Keeping directory: f:\temp\Base53383.tmp for module: x_client.jar.
Parsing XML file:f:\temp\Base53383.tmp\WarDeploy.wsdl
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java.
Done processing module x_client.jar.
```

Messages

- Flag *-f* is not valid.

  Option *f* was not recognized as a valid option.
- Flag *-c* is ambiguous.

  Options can be abbreviated, but the abbreviation must be unique. In this case, the **wsdeploy** command cannot determine which option was intended.
- Flag *-c* is missing parameter *-p*.

  A required parameter for an option is omitted.
- Missing *p* parameter.

  A required option is omitted.

Configuring Web service client bindings

When a Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise JavaBean (EJB) module, including client bindings.

Deploy the Web service into WebSphere Application Server.

To complete this task, you need to know the topology of the URL endpoint address of the Web services servers and which Web service the client depends upon. You can view the deployment descriptors in the administrative console to find topology information. See the article View Web services server deployment descriptors for more information.

The client bindings define the Web Services Description Language (WSDL) file name and preferred ports. The relative path of a Web service in a module is specified within a compatible WSDL file that contains the actual URL to be used for requests. The address is only needed if the original WSDL file did not contain a URL, or when a different address is needed. For a service endpoint with multiple ports, you need to define an alternative WSDL file name.

The following steps describe how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you must configure the client bindings to access the downstream Web service.

You can also configure client bindings with **wsadmin**.

To configure client bindings through the administrative console:

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* > **Web services client bindings**.
3. Find the Web service you want to update.

   The Web services are listed in the **Web Service** field.
4. Select the WSDL file name from the drop down box in the WSDL file name field.
5. Click **Edit** in the Preferred port mappings field to configure the default port to use.
   a. Specify the port type and the preferred ports in the Port type and Preferred ports fields.

      Configuring the preferred port enables you to select an optimal port implementation use non-SOAP protocols. See RMI-IIOP Web services using JAX-RPC for more information about using non-SOAP protocols.
   b. Click **Apply** and **OK**.
6. Click **Edit** in the Port information field to configure the request timeout, the overridden endpoint, and the overridden binding namespace for a port.

   Configuring the request timeout accommodates complex topologies that can have multiple cascaded Web services that involve multiple hops or long-running services.

Timeout values can be configured based on observed behavior of the overall system as integration proceeds. For example, a Web service client might time out because of changing network conditions or the performance of an external Web service. When you have applications containing Web services clients that timeout, you can change the request time out values for the clients.

a. Click **Apply** and **OK**.

Your Web service client bindings are configured.

Now you can finish any other configurations, start or restart the application, and verify the expected behavior of the Web service.

Web services client bindings

The client bindings define the Web Service Description Language (WSDL) file name, preferred ports and other port information. Use this page to specify the client bindings and the port mappings for the Web services in a module.

A Web service can specify the relative path within the module of a compatible WSDL file containing the actual URL to be used for requests. The relative path only needs to be specified if the original WSDL file does not contain a URL or when a different URL is needed. For a service endpoint with multiple ports defined, a preferred mapping specifies the default port to use for a port type.

Web service

Identifies the name of this Web service. A module can contain one or more Web services.

EJB

Identifies the name of the EJB for the EJB modules.

WSDL file name

Specifies the WSDL file name, which is relative to the module. Locate the WSDL file name in the drop down menu.

Preferred port mappings

Specifies and manages the preferred port type mapping for a Web service when a particular port type is requested.

Click **Edit** to edit the preferred port mapping information on the **Preferred port mappings** panel.

Port information

Specifies additional configuration information for the ports of this Web service.

Click **Edit** to edit the port information on the **Port information** panel. You can set a request timeout, override an endpoint and override a binding namespace for each client port.

Preferred port mappings

Use this page to view and manage a preferred portType mapping for a Web service.

When you have multiple ports that reference the same portType (service endpoint interface), a preferred port specifies the port to use when the `Service.getPort(Class SEI)` method is called with only the service endpoint interface.

To view this administrative console page, click **Applications** >**Enterprise Application** > *application_instance* > **Manage Modules** > *module_instance* >**Web services client bindings** > **Edit** > *preferred_port_instance*.

portType:

Specifies the portType.

The preferred port and the portType values are both of the type `java.xml.namespace.QName`.

Preferred port:

Specifies the preferred port to be associated with a particular portType. The `Service.getPort(Class)` method returns the preferred port associated with the specified service endpoint interface class (portType).

The preferred ports available are listed, as well as a value of `None`, which indicates no preferred port is selected.

Web services client port information

Use this page to specify a request timeout, override an endpoint, and override a binding namespace for a Web services client port.

A Web service can have multiple ports. You can view and configure the port attributes for each defined Web service port. The Web services are listed on the Web services client bindings panel.

To view this page, click **Applications** >**Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* >**Web services client bindings** > **Edit**.

For EJB modules, click **Applications** >**Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* >**Web services client bindings** > **Edit**.

Port:

Specifies the name of a port.

Request timeout:

Specifies the time, in seconds, that a Web service client waits for a request to complete on this port. If a timeout is not specified, the default request timeout for the client to wait is `360` seconds. If the value is set at `0` (zero), the client's request does not timeout.

A typical use for this setting is to customize the client's behavior when it is configured to use a JMS transport to access a Web service to make it wait longer for an expected completion. Depending upon network conditions, or the nature of a Web service implementation, it might be necessary to tune the timeout.

Overridden endpoint:

Specifies the name of an endpoint that is used to override the current endpoint. A client invoking a request on this port uses this endpoint instead of the endpoint specified in the WSDL file.

If an assembled application contains a Web service client that is statically bound, the client is locked into using the implementation (service end point) identified in the WSDL file used during development. Overriding the endpoint is an alternative to configuring the deployed WSDL attribute.

The overridden endpoint URI attribute is specified on a per port basis. It does not require an alternative WSDL file within the module. The overridden endpoint URI takes precedence over the deployed WSDL attribute. The client uses this value for the service end point URI or SOAP address, instead of the value in the static client bindings.

Overridden binding:

Specifies the WSDL file binding namespace URI to use with this port, instead of the namespace in the WSDL file. This binding does not need to exist in the WSDL file. A client invoking a request on this port uses this binding instead of the binding specified in the WSDL file. An overridden binding namespace cannot be specified unless an overridden endpoint is specified.

Getting started with the UDDI registry

This section covers the basic knowledge you need to get started either as an administrator of a UDDI registry or as a user of a UDDI registry that has already been set up.

- Getting started for UDDI Administrators
- Getting started for UDDI users

Getting started for UDDI Administrators

Use this topic if you are involved in installing (setting up and deploying), customizing, or managing a UDDI registry.

This section contains a list of some of the topics that you will need to refer to as an administrator of a UDDI registry:

- UDDI registry Terminology introduces some terms with which you will need to be familiar in order to administer a UDDI registry
- Setting up and deploying a new UDDI registry explains how to install a UDDI registry node by setting up the resources that it will use, and deploying the UDDI registry application.
- Migrating the UDDI registry explains how to use the scripts provided to migrate UDDI registries from the previous version of WebSphere Application Server.
- Migrating to Version 3 of the UDDI registry explains how to migrate UDDI Version 2 registries to UDDI Version 3, introduced in WebSphere Application Server Version 6.0, by creating a migration datasource.
- “Configuring UDDI registry security” on page 507 explains how to configure different levels of security for the UDDI registry, and how other UDDI node settings can influence security.
- Managing the UDDI registry explains how to use the UDDI pages in the administrative console, or the UDDI registry Administrative interface, to administer a UDDI registry node. It also covers how to back up and restore your UDDI registry data.
- UDDI node settings explains how to view and set UDDI properties and policies, and how to manage UDDI publishers, tiers and user entitlements.
- UDDI registry Management Interfaces covers details of programmatic interfaces that you can use to administer a UDDI registry node (UDDI registry Administrative (JMX) Interface), to add custom Value Set data to a UDDI registry (User Defined Value Set Support), and to export and import UDDI version 2 entities (UDDI Utility Tools).
- “Removing and reinstalling the UDDI registry” on page 503

UDDI registry troubleshooting might be useful if you encounter any problems or unexpected behavior while using the UDDI registry, and Troubleshooter reference: Messages links to messages that you might see.

Getting started for UDDI users

Use this topic if you use a UDDI registry to publish or find UDDI entities either through a user interface or by writing UDDI client applications.

This section contains a list of some of the topics that you might want to refer to as a user of a UDDI registry:

- UDDI registry Terminology introduces some terms with which you might need to be familiar in order to use a UDDI registry.

- Using the UDDI registry user interface explains how to access the UDDI User Console, which is a user interface that allows you to find UDDI entities and carry out simple UDDI publish operations.
- UDDI registry SOAP Service End Points contains details for accessing the UDDI version 3 Inquiry, Publish, Security, and custody transfer APIs, as well as the UDDI version 1 and version 2 APIs.
- UDDI registry Client Programming explains how to write UDDI client application programs. The recommended client programming interface is the UDDI version 3 Client for Java.
- IBM JAXR Provider for the UDDI registry is for users who want to use the Java API for XML Registries to access UDDI.
- User Defined Value Set Support in the UDDI registry explains how to add custom value set data to a UDDI registry.

UDDI registry troubleshooting might be useful if you encounter any problems or unexpected behavior while using the UDDI registry, and Troubleshooter reference: Messages links to messages that you might see.

Using the UDDI registry user interface

Configure the application server hosting the UDDI registry for UTF-8 encoding support, as described in refer to Configuring application servers for UTF-8 encoding.

This topic describes the UDDI registry user interface (also referred to as the UDDI registry user console), which you can use to explore the UDDI registry.

The user console provides a graphical user interface to the majority of the UDDI Version 3 API. It is not intended to support the full API set. There is some focus on inquiry operations, as the main purpose of the UDDI user console is to allow you to issue inquiry requests and to familiarize yourself with general UDDI concepts. This section documents the areas for which support through the user console is not provided, and other known restrictions to the user console.

- General
  - Help is provided in the form of explanatory text on the screens.
  - Maximum rows cannot be specified on finds. The single maximum rows value for the registry can be set through the *Maximum inquiry result set size* general property on the administrative console.
- Find business
  - The identifier feature is not supported.
- Find technical model (tModel)
  - The identifier feature is not supported.
- Add business
  - There is no support for adding Discovery URLs, Identifiers or Digital Signatures.
- Add technical model (tModel)
  - There is no support for adding Identifiers or Digital Signatures.
- Business Relationships
  - There is no support for Business Relationships

Use the related links below to find the appropriate topic for the task you want to perform using the UDDI registry user interface.

Displaying the UDDI registry user interface

Use this task to perform actions on UDDI information through the UDDI user console. The exact behavior of the user console depends on several configurable factors, such as:

- Whether WebSphere Application Server security is enabled.
- How the UDDI registry GUI role mappings are set. The UDDI registry supports a number of security roles, including two for the user console: GUI_Publish_User and GUI_Inquiry_User.
- How the UDDI registry GUI SSL transport guarantee constraints are set. The UDDI registry allows SSL settings to be configured and this includes two settings for the user console.

The following table summarizes the behavior of the UDDI registry user console.

Table 6.

WebSphere Application Server security status	URL used to access the UDDI user console	Behavior of the UDDI user console
Enabled	http://*host_name*:*http_port*/uddigui	Inquiry requests do not require authentication; they use the HTTP URL and are not secure. Publish requests do require WebSphere Application Server authentication. When you access the publish pane you will be dynamically redirected to use HTTPS, and will be prompted for a user ID and password. For the request to be successful , the authenticated user must be registered as a UDDI publisher. If the GUI_Inquiry_User role is mapped to all authenticated users, and the transport guarantee in the user data constraint section for that role is set to CONFIDENTIAL, *all* requests, including inquiry, require authentication and use of HTTPS.
	https://*host_name*:*ssl_port*/uddigui	Requests are secure; you are prompted to authenticate with a user ID and password. For the request to be successful , the authenticated user must be registered as a UDDI publisher.
Disabled	http://*host_name*:*http_port*/uddigui	No requests, either publish or inquire, are authenticated and the data flow is *not* secure (non SSL). Even though SSL transport-guarantee settings are defined, they are not enforced if security is disabled. All publish operations are performed using a user ID of UNAUTHENTICATED or a value that can be configured using the administrative console or the JMX management interface (this applies to new requests only).
	https://*host_name*:*ssl_port*/uddigui	No requests, either publish or inquire, are authenticated, but the data flow is secure because the SSL URL and port are used explicitly. All publish operations are performed using a user ID of UNAUTHENTICATED or a value that can be configured using the administrative console or the JMX management interface (this applies to new requests only).

The variables in the table have the following values:

- *host_name* is the name of the machine that is running the relevant profile.
- *http_port* is the internal HTTP port for the profile, for example 9080.
- *ssl_port* is the internal SSL port for the profile, for example 9443.

1. Start the UDDI application, if it is not already running.
2. Open a browser window and ensure that cookies are enabled.
3. Access the UDDI registry user console using one of the following default URLs.
   - http://*host_name*:*http_port*/uddigui
   - https://*host_name*:*ssl_port*/uddigui

The user console displays the default frameset containing the following items:

- The header frame.
- The navigation frame showing find options.
- The details frame.

You can now use the UDDI user console to find, edit or publish UDDI information.

Finding an entity using the UDDI registry user interface

Make sure that the UDDI registry application is started, then display the UDDI registry user interface as described in Displaying the user interface.

This topic describes how to use the UDDI registry user interface (also referred to as the UDDI registry user console) to find services, businesses and technical models.

1. Activate the **Find** tab by clicking it, or by clicking the **Find** link at the top of the page or on the Welcome page.
2. To perform a quick find, complete the following steps:
   a. In the **Quick Find** section of the **Find** tab, select the kind of entity you want to find; service, business or technical model.
   b. In the **Starting with** field, enter name of the entity. Use the '%' wildcard character to search for a partial name.
   c. Click **Find**.

   The results are displayed in the detail frame on the right.
3. To perform an advanced find, complete the following steps:
   a. In the **Advanced Find** section of the **Find** tab, click the appropriate link for the kind of entity you want to find; service, business or technical model. The advanced search form is displayed in the frame to the right.
   b. Enter your search criteria in the advanced search form, and select any find qualifiers you require. You must enter at least one name to search for, using the **Add Name** link. You can use this link to enter multiple names. You can also add multiple categorizations. To add a categorization, use the **Show category tree** link in the **Categorizations** section to display, in the pane on the left, a tree of categories (or taxonomies) defining the types of item to find according to various classification systems. Expand the tree to find the category that you want, click the category to add the information to the advanced search form, then use the **Add Categorization** link to include the category in the search.
   c. Click **Find *entities***.

   The results are displayed in the detail frame on the right.

Publishing an entity using the UDDI registry user interface

Make sure that the UDDI registry application is started, then display the UDDI registry user interface as described in Displaying the user interface.

This topic describes how to use the UDDI registry user interface (also referred to as the UDDI registry user console) to publish businesses and technical models. To publish a service, first publish a business and then add a service to that business, as described in Editing or deleting an entity using the UDDI registry user interface.

1. Activate the **Publish** tab by clicking it, or by clicking the **Publish** link at the top of the page or on the Welcome page.
2. To publish an entity by name only, use the quick publish section as follows:
   a. In the **Quick Publish** section of the **Publish** tab, select the kind of entity you want to publish; business or technical model.
   b. In the **Name** field, enter name of the entity.
   c. Click **Publish**.

   The details of the published entity are displayed in the frame on the right.
3. To publish an entity with more information, complete the following steps:

a. In the **Advanced Publish** section of the **Publish** tab, click the appropriate link for the kind of entity you want to publish; business or technical model. The advanced publish form is displayed in the frame to the right.
b. Enter the details for the entity in the advanced publish form. You can enter multiple names, descriptions, contacts or categorizations by using the relevant **Add** link. To add a categorization, first use the **Show category tree** link in the **Categorizations** section to display, in the pane on the left, a tree of categories (or taxonomies) defining the types of item to publish according to various classification systems. Expand the tree to find the category that you want, click the category to add the information to the advanced publish form, then click the **Add Categorization** link.
c. Click **Publish *entity*** to publish the business or technical model to the UDDI registry.

Editing or deleting an entity using the UDDI registry user interface

Make sure that the UDDI registry application is started, then display the UDDI registry user interface as described in Displaying the user interface.

This topic describes how to use the UDDI registry user interface (also referred to as the UDDI registry user console) to edit or delete the businesses and technical models that you own, including adding services to businesses.

1. Activate the **Publish** tab by clicking it, or by clicking the **Publish** link at the top of the page or on the Welcome page.
2. At the bottom of the **Publish** tab is the **Registered Information** section. Click **Show owned entities** to show the businesses and technical models that you have registered in the UDDI registry.
3. **Optional:** To delete a business or technical model, click the **Delete** link in the **Actions** column for that entity.

   Note: When you delete a technical model, it is hidden rather than physically deleted, as specified by the UDDI Version 3.0 specification. If you click the **Shown owned entities** link the technical model will still appear, but you will not be able to find it using the **Find** function. All other entities are deleted from the UDDI registry in the normal way.
4. **Optional:** To edit a business or technical model, click the **Edit** link in the **Actions** column for that entity, fill in the required details and click **Update *entity*** to save the changes in the UDDI registry.
5. **Optional:** To add a service to a business, click the **Add service** link in the **Actions** column for the business. Fill in the details and click **Add Service** to publish the service to the UDDI registry. The service details are displayed.

Creating business relationships using the UDDI registry user interface

If your business has an association with another business in the UDDI registry, for example a preferred supplier, you can describe this association in the UDDI registry by creating a *business relationship*.

1. The UDDI registry contains the following default relationship types:

   Parent-child
   A hierarchical relationship exists between the two business entities, which might represent, for example, a large organization and a subsidiary.

   Peer-peer
   The two business entities represent peer organizations, for example a company and its supplier.

   Identity
   The two business entities represent the same organization.

   If you require a different relationship type, create a user-defined value set to represent the relationship type that you require, as described in “User-defined value set support in the UDDI registry” on page 478.

2. Each business that is involved in the relationship must already exist in the UDDI registry.
3. Make sure that the UDDI registry application is started, then display the UDDI registry user interface as described in Displaying the user interface.

Perform this task when you want to publicize an association between two businesses in the UDDI registry. For example, your organization, represented by a business entity in the UDDI registry, might have several departments, each one represented by a different business entity in the UDDI registry. You might want to declare these departments as being linked to the parent organization, by creating parent-child relationships between the appropriate business entities.

1. Activate the **Publish** tab by clicking it, or by clicking the **Publish** link at the top of the page or on the Welcome page.
2. Under **Registered Information**, click **Show owned entities**. The entities that you own are displayed in the detail frame on the right.
3. In the section for the businesses that you own, find the business that you want to link from, and click the **Add relationship** link for that business. The Add Business Relationship pane appears. The business key for the business that you selected is already listed in the From section.
4. Click **Add** to add the second business, the business that you want to link to. The advanced find pane appears.
5. Find the second business, as described in the advanced find step of “Finding an entity using the UDDI registry user interface” on page 399.
6. Click **Select** to add the second business to the relationship. If you want to change the positions of the businesses, click **Swap**.
7. Select the type of relationship from the **Type** list.
8. If required, type a description in the **Usage** field.
9. Click **Add relationship** to create the relationship. If you own both businesses, no further action is required. The relationship appears as a *publisher assertion* in the list of entities that you own. The status of the assertion is complete.
10. If you do not own the second business, the status of the assertion is pending. The owner of the second business must create a relationship from their business to yours, for the relationship to be complete and visible to other parties. The relationship type must match the type that you chose earlier.

The business relationship is published to the UDDI registry. The UDDI user interface only shows publisher assertions for entities that you own. To view other relationships, use the UDDI inquiry API provided.

For more information about publisher assertions, refer to the UDDI specification.

To remove an assertion that you own, display your owned entities and click the **Delete** link for the relevant publisher assertion. If the business in the To field is owned by someone else, the status of the assertion becomes pending, and the relationship is no longer visible to other parties.

Example: Publishing a business, service and technical model using the UDDI registry user interface

This example describes how to use the UDDI registry user interface to publish a used car business called Modern Cars to the UDDI registry, and how to publish a service and technical model for the business.

Before you begin, make sure that the UDDI registry application is started, then display the UDDI user interface as described in Displaying the user interface.

Adding the Business

1. Click the **Publish** tab to activate the **Publish** pane.
2. Under **Advanced Publish** click **Add a business**. The advanced publish form is displayed on the right.

3. Type 'Modern Cars' in the **Name** field for the business. Select the language of the business name from the drop down list, then click **Add Name** to add the name to the business.
4. Type a description for the business, such as 'Used cars for sale' in the **Description** field. Select the language of the description from the drop down list, then click **Add Description** to add the description to the business. You can add multiple descriptions in a variety of languages as required.
5. In the **Contact** section, type your name as a contact for customers of the business. Select the language as before, and click **Add Contact**. The business contact form is displayed. Fill in the details, using the **Add *entity*** links to add the information as you reach the end of each subsection. Note that all the text fields in the form are cleared when you click an **Add *entity*** link. Click **Add Contact** to save the contact information into the Modern Cars business.
6. Use the **Categorizations** section to describe the Modern Cars business according to the NAICS 2002 categorization system:
   a. Click **Show category tree** to display the various categorization systems.
   b. Expand the **NAICS 2002** tree, then in that tree expand **Retail Trade [44]** → **Motor Vehicle and Parts Dealers [441]** → **automobile Dealers [4411]** → **Used Car Dealers [44112]**. Click the **Used Car Dealers [441120]** category to add the category type, key name and key value to the advanced publish form.
   c. Click **Add Categorization** to add the information to the business.
   d. Close the category tree by clicking the **close** link near the top of the tree.
7. Click **Publish Business** to publish the Modern Cars business to the UDDI registry. The details of the business will be displayed.

Adding a service to the business

1. Click the **Publish** tab to activate the **Publish** pane.
2. Under **Registered Information** click **Show owned entities** to display the Modern Cars business, and any other entities that are owned by you.
3. Click **Add service** in the **Actions** column of the Modern Cars business. The publish service page is displayed.
4. Add a name and description for the service, in the same way as for the business itself.
5. Click **Add a Service Binding** to display the service binding form. Enter an access point (the URL for the service on the network) and a description for the service binding. Click **Add Technical Model Instance Information** to display a page where you can describe and publish a technical model instance for the service binding. In the technical model information page, click **Add Technical Model**. Search for the technical model which is used by the instance, select the technical model from the results and click **Add**. Fill in the other fields on the form and click **Add Technical Model Instance**. Click **Add Binding** to save the information into the service.
6. Add a categorization as you did for the business itself.
7. Click **Add Service** to publish the service to the UDDI registry.

Adding a technical model

1. In the **Advanced Publish** section on the left, click **Add a technical model** to display the publish technical model form on the right.
2. Add a name and description for the technical model, in the same way as for the business and the service.
3. Click **Add an Overview Document** to display the overview document form. The overview document describes the technical model. Type the location of the overview document in the **Overview URL** field, and click **Add Overview Document URL**. Add a description and click **Add Overview Document** to save the information in the technical model.
4. Add a categorization in the same way as for the business.
5. Click **Publish Technical Model** to publish the technical model to the UDDI registry.

Planning to use Web services

This topic discusses how to plan your use of Web services that are developed and implemented based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

Read the Web services scenario: Overview which tells the story of a fictional online garden supply retailer named Plants by WebSphere and how this retailer incorporated the Web services concept. You can also review the Samples Gallery for Web services samples. These Samples demonstrate Enterprise JavaBeans (EJBs) and JavaBeans components that are available as Web services.

Web services reflect the service-oriented architecture approach to programming. This approach is based on the idea of building applications by discovering and implementing network-available services, or by invoking the available applications to accomplish a task. Web services deliver interoperability, for example, Web services applications provide a way for components created in different programming languages to work together as if they were created using the same language. Web services rely on existing transport technologies, such as HTTP, and standard data encoding techniques, such as Extensible Markup Language (XML), for invoking the implementation.

To plan to use Web services:

1. Identify your goals and design Web services to fit your e-business solution. Consider what you want to accomplish by using Web services. Decide how Web services fit into your current topology, applications and programming model. Determine how the Web services process requests on the server and how the clients manage and use the Web service.
2. Design your Web services for reliability, availability, manageability and security. For example, you want your Web services to process a transaction in a reasonable time at all hours of the day and provide users with good security characteristics, such as authentication for buyers. Planning to use Web services to work with WebSphere Application Server helps to meet these requirements.
3. Review the standards used in developing and deploying Web services into WebSphere Application Server. Development and deployment are based on the J2EE and Java API for XML-based remote procedure call (JAX-RPC) programming models. There are extensions to these standards that are also important to review. See Extensions to the JAX-RPC and Web Services for J2EE programming models for more information.
4. Decide what development and implementation tools to use. You can use a variety of manual development and implementation tasks. Whether you have an existing Web service to implement or you want to develop your own from a JavaBeans implementation or from an Enterprise JavaBeans (EJB) module, you can choose different tasks respective to your resources. You can also use Rational Application Developer (RAD) to complete development and implementation tasks.

   See Developing Web services for information about developing Web services based on the J2EE specification through the WebSphere Application Server administrative console and command-line tools. To read more about RAD see the information center for the product.
5. Install WebSphere Application Server.

   See Install WebSphere Application Server.
6. Review Web services Samples.

You have a design plan for implementing Web services applications into your business architecture.

Develop a Web service.

This topic explains how to develop a Web service using the Web Services for J2EE specification.

Service-oriented architecture

A *service-oriented architecture* (SOA) is a collection of services that communicate with each other, for example, passing data from one service to another or coordinating an activity between one or more services.

Companies want to integrate existing systems to implement Information Technology (IT) support for business processes that cover the entire business value chain. A variety of designs are used, ranging from rigid point-to-point electronic data interchange (EDI) to Web auctions. By using the Internet, companies can make their IT systems available to internal departments or external customers, but the interactions are not flexible and are without standardized architecture.

Because of this increasing demand for technologies that support connecting and sharing resources and data, a need exists for a flexible, standardized architecture. SOA is a flexible architecture that unifies business processes by structuring large applications into building blocks, or small modular functional units or services, for different groups of people to use inside and outside the company. The building blocks can be one of three roles: service provider, service broker, or service requestor. See Web services approach to a service-oriented architecture to learn more about these roles.

Requirements for an SOA

To efficiently use an SOA, follow these requirements:

- **Interoperability between different systems and programming languages**.

  The most important basis for a simple integration between applications on different platforms is to provide a communication protocol. This protocol is available for most systems and programming languages.
- **Clear and unambiguous description language**.

  To use a service offered by a provider, it is not only necessary to be able to access the provider system, but the syntax of the service interface must also be clearly defined in a platform-independent fashion.
- **Retrieval of the service**.

  To support a convenient integration at design time or even system run time, a search mechanism is required to retrieve suitable services. Classify these services as *computer-accessible*, *hierarchical* or *taxonomies* based on what the services in each category do and how they can be invoked.

Web services approach to a service-oriented architecture

This article describes how Web services are used in a service-oriented architecture (SOA).

You can use Web services to implement a SOA. A major focus of Web services is to make functional building blocks accessible over standard Internet protocols that are independent from platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network-enabled. A service can rely on another service to achieve its goals.

Each SOA building block can assume one or more of three roles:

- **Service provider**

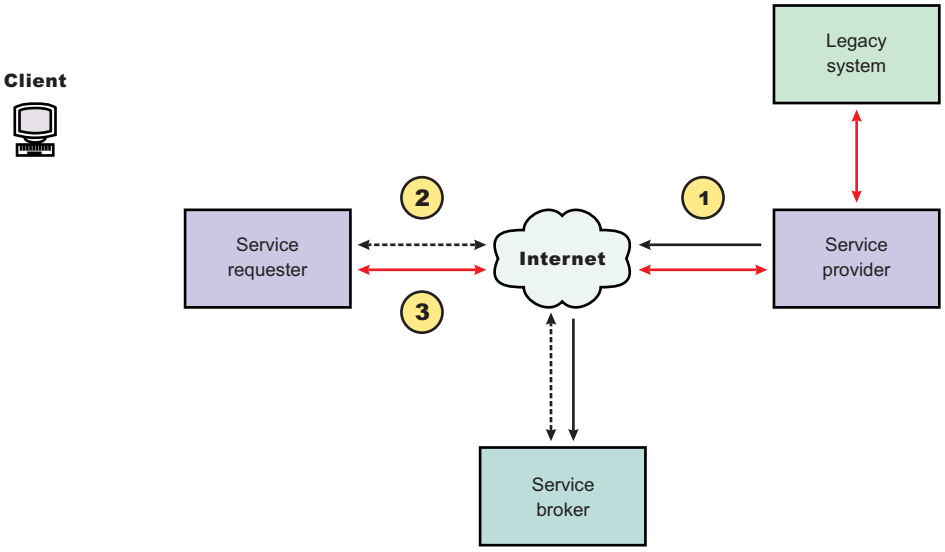
  The service provider creates a Web service and possibly publishes its interface and access information to the service registry. Each provider must decide which services to expose, how to make trade-offs between security and easy availability, how to price the services, or how to exploit free services for other value. The provider also has to decide which category to list the service in for a given broker service and what sort of trading partner agreements are required to use the service.
- **Service broker**

  The service broker, also known as *service registry*, is responsible for making the Web service interface and implementation access information available to any potential service requestor. The implementer of the broker decides the scope of the broker. Public brokers are available through the Internet, while

private brokers are only accessible to a limited audience, for example, users of a company intranet. Furthermore, some decisions need to be made about the amount of the offered information. Some brokers specialize in many listings. Others offer high levels of trust in the listed services. Some cover a broad landscape of services and others focus within an industry. Some brokers catalog other brokers. Depending on the business model, brokers can attempt to maximize look-up requests, the number of listings or the accuracy of the listings. The Universal Description, Discovery and Integration (UDDI) specification defines a way to publish and discover information about Web services.

- **Service requester**

  The service requestor or Web service client locates entries in the broker registry using various find operations and then binds to the service provider to invoke one of its Web services.



Characteristics of the SOA

The presented SOA illustrates a loose coupling between the participants, which provides greater flexibility in the following ways:

- A client is coupled to a service. Therefore, the integration of the server takes place outside the scope of the client application programs.
- Old and new functional blocks or applications and systems, are encapsulated into components that work as services.
- Functional components and their interfaces are separate so that new interfaces can be plugged in more easily.
- Within complex applications, the control of business processes can be isolated. A business rule engine can be incorporated to control the workflow of a defined business process. Depending on the state of the workflow, the engine calls the respective services.
- Services can be incorporated dynamically during run time.
- Bindings are specified using configuration files and can be easily adapted to new needs.

Properties of a service-oriented architecture

The service-oriented architecture offers the following properties:

- **Web services are self-contained**

  On the client side, no additional software is required. A programming language with Extensible Markup Language (XML) and HTTP client support is enough to get you started. On the server side, a Web server and a SOAP server are required. It is possible to enable an existing application for Web services without writing a single line of code.

- **Web services are self-describing**

  Neither the client nor the server knows or cares about anything besides the format and content of the request and response messages (loosely coupled application integration). The definition of the message format travels with the message; no external metadata repositories or code generation tool are required.
- **Web services can be published, located, and invoked across the Internet**

  This technology uses established lightweight Internet standards such as HTTP and it leverages the existing infrastructure. Some other standards that are required include, SOAP, Web Services Description Language (WSDL), and UDDI.
- **Web services are language-independent and interoperable**

  The client and server can be implemented in different environments. Existing code does not have to change in order to be Web services-enabled.
- **Web services are inherently open and standard-based**

  XML and HTTP are the major technical foundation for Web services. A large part of the Web service technology has been built using open-source projects.
- **Web services are dynamic**

  Dynamic e-business can become reality using Web services because with UDDI and WSDL you can automate the Web service description and discovery.
- **Web services are composable**

  Simple Web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer Web services from a Web service implementation. Web services can be chained together to perform higher-level business functions. This chaining shortens development time and enables best-of-breed implementations.
- **Web services are loosely coupled**

  Traditionally, application design has depended on tight interconnections at both ends. Web services require a simpler level of coordination that supports a more flexible reconfiguration for an integration of the services.
- **Web services provide programmatic access**

  The approach provides no graphical user interface; it operates at the code level. Service consumers need to know the interfaces to Web services, but do not need to know the implementation details of services.
- **Web services provide the ability to wrap existing applications**

  Already existing stand-alone applications can easily integrate into the SOA by implementing a Web service as an interface.

Web services business models supported

The properties and benefits of using a service-oriented architecture (SOA) such as Web services is well suited for binding small modules that perform independent tasks within a highly heterogeneous e-business model. Web services can be easily wrapped around existing applications in your business model and plugged into different business processes.

For connecting to a large monolithic system that does not support the implementation of different flexible business processes, other approaches might be better suited, for example, to satisfy specialized features, such as performance or security.

The following business models are easily implemented by using an architecture including Web services:

- **Business information**

  Sharing of information with consumers or other businesses. Web services can be used to expand the reach through such services as news streams, local weather reports, integrated travel planning, and intelligent agents.
- **Business integration**

Providing transactional, fee-based services for customers. A global network of suppliers can be easily created. Web services can be implemented in auctions, e-marketplaces, and reservation systems.

- **Business process externalization**

  Web services can be used to model value chains by dynamically integrating processes to a new solution within an organizational unit or even with those of other e-businesses. This modeling can be achieved by dynamically linking internal applications to new partners and suppliers, to offer their services to complement internal services.

To see how these models are implemented using all aspects of Web services, see Web services scenario: Overview which tells the story of a fictional online garden supply retailer named Plants by WebSphere and how this retailer incorporates the Web services concept.

Learning about the Web Services Invocation Framework (WSIF)

The Web Services Invocation Framework (WSIF) is a WSDL-oriented Java API. You use this API to invoke Web services dynamically, regardless of the service implementation format (for example enterprise bean) or the service access mechanism (for example Java Message Service (JMS)).

Using WSIF, you can move away from the usual Web services programming model of working directly with the SOAP APIs, towards a model where you interact with representations of the services. You can therefore work with the same programming model regardless of how the service is implemented and accessed.

To learn about WSIF, see the following topics:

- "Goals of WSIF."
  1. "WSIF - Web services are more than just SOAP services" on page 408.
  2. "WSIF - Tying client code to a particular protocol implementation is restricting" on page 408.
  3. "WSIF - Incorporating new bindings into client code is hard" on page 408.
  4. "WSIF - Multiple bindings can be used in flexible ways" on page 408.
  5. "WSIF - Enabling a freer Web services environment promotes intermediaries" on page 409.
- "WSIF: Overview" on page 409.
  1. "WSIF architecture" on page 409.
  2. "WSIF and Web services that offer multiple bindings" on page 410.
  3. "WSIF and WSDL" on page 410.
  4. "WSIF usage scenarios" on page 411.
  5. "Dynamic invocation" on page 412.

For more information about working with WSIF, visit the Web sites listed in Web services: Resources for Learning.

Related reference

Web services: Resources for Learning
This topic provides relevant supplemental information about Web services-related topics.

Goals of WSIF

WSIF aims to extend the flexibility provided by SOAP services into a general model for invoking Web services, irrespective of the underlying binding or access protocols.

SOAP bindings for Web services are part of the WSDL specification, therefore when most developers think of using a Web service, they immediately think of assembling a SOAP message and sending it across the network to the service endpoint, using a SOAP client API. For example: using Apache SOAP the client

creates and populates a Call object that encapsulates the service endpoint, the identification of the SOAP operation to invoke, the parameters to send, and so on.

While this process works for SOAP, it is limited in its use as a general model for invoking Web services for the following reasons:
- Web services are more than just SOAP services.
- Tying client code to a particular protocol implementation is restricting.
- Incorporating new bindings into client code is hard.
- Multiple bindings can be used in flexible ways.
- A freer Web services environment enables intermediaries.

The goals of the Web Services Invocation Framework (WSIF) are therefore:
- To give a binding-independent mechanism for Web service invocation.
- To free client code from the complexities of any particular protocol used to access a Web service.
- To enable dynamic selection between multiple bindings to a Web service.
- To help the development of Web service intermediaries.

WSIF - Web services are more than just SOAP services

You can deploy as a Web service any application that has a WSDL-based description of its functional aspects and access protocols. If you are using the Java 2 platform, Enterprise Edition (J2EE) environment, then the application is available over multiple transports and protocols.

For example, you can take a database-stored procedure, expose it as a stateless session bean, then deploy it into a SOAP router as a SOAP service. At each stage, the fundamental service is the same. All that changes is the access mechanism: from Java DataBase Connectivity (JDBC) to Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) and then to SOAP.

The WSDL specification defines a SOAP binding for Web services, but you can add binding extensions to the WSDL so that, for example, you can offer an enterprise bean as a Web service using RMI-IIOP as the access protocol. You can even treat a single Java class as a Web service, with in-thread Java method invocations as the access protocol. With this broader definition of a Web service, you need a binding-independent mechanism for service invocation.

WSIF - Tying client code to a particular protocol implementation is restricting

If your client code is tightly bound to a client library for a particular protocol implementation, it can become hard to maintain.

For example, if you move from Apache SOAP to Java Message Service (JMS) or enterprise bean, the process can take a lot of time and effort. To avoid these problems, you need a protocol implementation-independent mechanism for service invocation.

WSIF - Incorporating new bindings into client code is hard

If you want to make an application that uses a custom protocol work as a Web service, you can add extensibility elements to WSDL to define the new bindings. But in practice, achieving this capability is hard.

For example you have to design the client APIs to use this protocol. If your application uses just the abstract interface of the Web service, you have to write tools to generate the stubs that enable an abstraction layer. These tasks can take a lot of time and effort. What you need is a service invocation mechanism that allows you to update existing bindings, and to add new bindings.

WSIF - Multiple bindings can be used in flexible ways

To take advantage of Web services that offer multiple bindings, you need a service invocation mechanism that can switch between the available service bindings at run time, without having to generate or recompile a stub.

Imagine that you have successfully deployed an application that uses a Web service which offers multiple bindings. For example, imagine that you have a SOAP binding for the service and a local Java binding that lets you treat the local service implementation (a Java class) as a Web service.

The local Java binding for the service can only be used if the client is deployed in the same environment as the service. In this case, it is more efficient to communicate with the service by making direct Java calls than by using the SOAP binding.

If your clients could switch the actual binding used based on run-time information, they could choose the most efficient available binding for each situation.

WSIF - Enabling a freer Web services environment promotes intermediaries

Web services offer application integrators a loosely-coupled paradigm. In such environments, intermediaries can be very powerful.

Intermediaries are applications that intercept the messages that flow between a service requester and a target Web service, and perform some mediating task (for example logging, high-availability or transformation) before passing on the message. The Web Services Invocation Framework (WSIF) is designed to make building intermediaries both possible and simple. Using WSIF, intermediaries can add value to the service invocation without needing transport-specific programming.

WSIF: Overview

The Web Services Invocation Framework (WSIF) provides a Java API for invoking Web services, independent of the format of the service or the transport protocol through which it is invoked.

This framework addresses all of the issues identified in “Goals of WSIF” on page 407.

WSIF provides the following features:
- An API that provides binding-independent access to any Web service.
- A close relationship with WSDL, so it can invoke any service that you can describe in WSDL.
- A stubless and completely dynamic invocation of a Web service.
- The capability to plug a new or updated implementation of a binding into WSIF at run time.
- The option to defer the choice of a binding until run time.

WSIF is designed to work both in an unmanaged environment (stand-alone) and inside a managed container. You can use the Java Naming and Directory Interface (JNDI) to find the WSIF service, or you can use the location described in the WSDL.

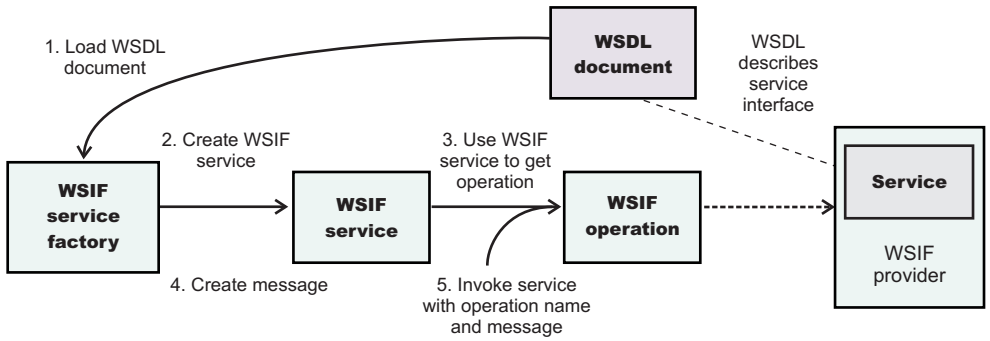
For more conceptual information about WSIF and WSDL, see the following topics:
- WSIF and WSDL
- WSIF architecture
- WSIF and Web services that offer multiple bindings
- WSIF usage scenarios
- Dynamic invocation

WSIF supports Internet Protocol Version 6, and Java API for XML-based Remote Procedure Calls (JAX-RPC) Version 1.1 for SOAP.

WSIF architecture

A diagram depicting the Web Services Invocation Framework (WSIF) architecture, and a description of each of the major components of the architecture.

The Web Services Invocation Framework (WSIF) architecture is shown in the figure.



The components of this architecture include:

WSDL document
: The Web service WSDL document contains the location of the Web service. The binding document defines the protocol and format for operations and messages defined by a particular portType.

WSIF service
: The WSIFService interface is responsible for generating an instance of the WSIFOperation interface to use for a particular invocation of a service operation. For more information, see Finding a port factory or service

WSIF operation
: The run-time representation of an operation, called *WSIFOperation* is responsible for invoking a service based on a particular binding. For more information, see WSIF API reference: Using ports.

WSIF provider
: A WSIF provider is an implementation of a WSDL binding that can run a WSDL operation through a binding-specific protocol. WSIF includes SOAP providers, JMS providers, Java providers and EJB providers. For more information, see Linking a WSIF service to the underlying implementation of the service.

WSIF and Web services that offer multiple bindings

Using WSIF, a client application can choose dynamically the optimal binding to use for invoking Web service operations.

For example, a Web service might offer a SOAP binding, and also a local Java binding so that you can treat the local service implementation (a Java class) as a Web service. If a client application is deployed in the same environment as the service, then this client can use the local Java binding for the service. This provides more efficient communication between the client and the service by making direct Java calls rather than indirect calls using the SOAP binding.

For more information about how to configure a client to dynamically select between multiple bindings, see Developing a WSIF service.

WSIF and WSDL

There is a close relationship between the metadata-based Web Services Invocation Framework (WSIF) and the evolving semantics of Web Services Description Language (WSDL).

In WSDL, a service is defined in three distinct sections:

- The **portType**. This section defines the abstract interface offered by the service. A portType defines a set of *operations*. Each operation can be In-Out (request-response), In-Only, Out-Only and Out-In (Solicit-Response). Each operation defines the input and/or output *messages*. A message is defined as a set of *parts*, and each part has a schema-defined type.

- The **binding**. This section defines how to map between the abstract portType and a real service format and protocol. For example the SOAP binding defines the encoding style, the SOAPAction header, the namespace of the body (the targetURI), and so on.
- The **port**. This section defines the actual location (endpoint) of the available service. For example, the HTTP Web address at which a SOAP service is available.

Currently in WSDL, each port has one and only one binding, and each binding has a single portType. But (more importantly) each service (portType) can have multiple ports, each of which represents an alternative location and binding for accessing that service.

The Web Services Invocation Framework (WSIF) follows the semantics of WSDL as much as possible:
- The WSIF dynamic invocation API directly exposes run-time equivalents of the model from WSDL. For example, invocation of an operation involves executing an operation with an input message.
- WSDL has extension points that support the addition of new ports and bindings. This enables WSDL to describe new systems. The equivalent concept in WSIF is a provider, that enables WSIF to understand a class of extensions and thereby to support a new service implementation type.

As a metadata-based invocation framework, WSIF follows the design of the metadata. As WSDL is extended, WSIF is updated to follow.

The implicit and primary type system of WSIF is XML schema. WSIF supports invocation using dynamic proxies, which in turn support Java type systems, but when you use the WSIFMessage interface it is your responsibility to populate WSIFMessage objects with data based on the XML schema types as defined in the WSDL document. You should define your object types by a canonical and fixed mapping from schema types into the run-time environment.

For more information about WSDL, see Web services: Resources for learning.

WSIF usage scenarios

This topic describes two brief scenarios that illustrate the role WSIF plays in the emerging Web services environment.

Scenario: Redevelopment and redeployment

When you first implement a Web service, you create a simple prototype. When you want to move a prototype Web service into production, you often need to redevelop and redeploy it.

The Web Services Invocation Framework (WSIF) uses the same API calls irrespective of the underlying technologies, therefore if you use WSIF:
- You can reimplement and redeploy your services without changing the client code.
- You can use existing reliable and high-performance infrastructures like Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) and Java Message Service (JMS) without sacrificing the location-independence that the Web service model offers.

Scenario: Service Flow composition

A service flow typically invokes a Web service, then passes the response from one Web service to the next Web service, perhaps performing some transformation in the middle.

There are two key aspects to this flow that WSIF provides:
- A representation of the service invocation based on the metadata in WSDL.
- The ability to build invocations based solely on the portType, which can therefore be used in any implementation.

For example, imagine that you build a meta-service that uses a number of services to build a process. Initially, several of those services are simple Java bean prototypes that are written and exposed through SOAP, but you plan to reimplement some of them as EJB components, and to out-source others.

If you use SOAP, it ties up multiple threads for every onward invocation, because they pass through the Web server and servlet engine and on to the SOAP router. If you use WSIF to call the beans directly, you get much better performance compared to SOAP and you do not lose access or location transparency. Using WSIF, you can replace the Java bean implementations with EJB implementations without changing the client code. To move some of the Web services from local implementations to external SOAP services, you just update the WSDL.

Dynamic invocation

The Web Services Invocation Framework (WSIF) can provide runtime support for Web services, and for WSDL extensions and bindings, that were not known at build time.

WSIF supports WSDL extensions and bindings that were not known at build time through the use of providers. The providers support Web services that were not known at build time by using the WSDL description to access the target service.

Setting up and deploying a new UDDI registry

Start WebSphere Application Server, and create a server to host the UDDI registry. Use Chapter 3, “Starting and stopping quick reference,” on page 11 for information about starting WebSphere Application Server using either commands or the administrative console.

A UDDI registry node consists of the UDDI registry application (a J2EE application that is supplied as part of WebSphere Application Server), a store of data (using a relational database management system) referred to as the UDDI database, and a means to connect the application to the data (a datasource and related elements). The ’Setting up’ sub-topics describe how to create the database (which can be local or remote) and datasource, and how to deploy the UDDI registry application.

You can create either a *default* UDDI node or a *customized* UDDI node. The main difference between default and customized, in the context of these set up tasks, refers to a number of mandatory UDDI registry properties such as the UDDI node ID and description, and the prefix to be used for generated discovery URLs.

Default UDDI node
: The mandatory properties are automatically set to default values and cannot be changed. A default UDDI node is a suitable option for initial evaluation of the UDDI registry, and for development and test purposes.

Customized UDDI node
: You must set the mandatory properties, but once set they cannot be changed for this configuration. With a customized UDDI node you have more control over the database management system used for the UDDI database, and the properties used to set up the UDDI database. With a customized UDDI node, you create the UDDI database and datasource to your own specifications before deploying the UDDI registry application. A customized node is a suitable option for production purposes. To move from a default UDDI node to a customized node, see “Changing the UDDI registry application environment after deployment” on page 441.

Proceed to one of the following topics:

- “Setting up a default UDDI node with a default datasource” on page 413. Use this topic if you want to quickly set up a UDDI registry for test or development purposes. The database, datasource and UDDI registry application are created or deployed by a single script. Note that the database type is embedded Cloudscape.

- “Setting up a default UDDI node” on page 414. Use this topic if you want to create a default UDDI registry with a database other than embedded Cloudscape, or if you want an embedded Cloudscape database but you want to create the datasource manually.
- “Setting up a customized UDDI node” on page 427

Database considerations for production use of the UDDI registry

The UDDI registry fully supports a number of databases (see An overview of the Version 3 UDDI registry for details) and can be used for development and test purposes, however you should be aware of the following factors when considering which database is appropriate for your anticipated UDDI registry production use.

It is important to consult the information supplied by your chosen database vendor for advice, but additionally you need to consider the likely size and volume of requests, and whether the general performance and scalability of the UDDI registry is important to you.

For example, while Cloudscape supports the full function of the UDDI registry, it is not an enterprise level database and consequently it does not have the same characteristics (for example, scaling or performance) as enterprise databases such as DB2.

If you need multiple connections to the UDDI registry database (for example if you are using the UDDI registry in a cluster configuration) and Cloudscape is your preferred database, you will need to use the network Cloudscape option as embedded Cloudscape has a limitation of allowing only one Java virtual machine to access or load a database instance at any one time (in other words two application servers cannot access the same Cloudscape database instance at the same time).

Note: The UDDI registry can support multiple users even if there is a single database connection.

More information on Cloudscape is available in this information center.

Setting up a default UDDI node with a default datasource

Use this task to create a UDDI node with predetermined property values and an embedded Cloudscape database. You will not be able to change the mandatory node properties, such as node ID, either during the creation of the node or afterwards. Such a node is suitable for initial evaluation of the UDDI registry and for development and test purposes. If you want to choose the mandatory node properties yourself, set up a customized node as detailed in “Setting up a customized UDDI node” on page 427.

The UDDI database and datasource are automatically created by running a single script, which also deploys the UDDI registry application. If you want to have more control over the datasource, refer to “Setting up a default UDDI node” on page 414.

1. Create the UDDI node by running the wsadmin script uddiDeploy.jacl from the *app_server_root*/bin directory. The syntax of the command is shown below.

   Note: If you are using either the UNIX or Linux operating systems, add the `.sh` suffix to the wsadmin command.

```
wsadmin [-conntype none] [-profileName profile_name] -wsadmin_classpath app_server_root/derby/lib
              -f uddiDeploy.jacl
              node_name
              server_name
              default
```

   where

   - ’-profileName *profile_name*’ is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
   - ’-conntype none’ is optional, and is only needed if the application server is not running.

- *app_server_root* is the directory name of the WebSphere Application Server installation location.
- *node_name* is the name of the WebSphere node on which the target server runs. Note that the node name is case sensitive.
- *server_name* is the name of the target server on which you wish to deploy the UDDI registry, such as server1. Note the server name entered is case sensitive.
- 'default' causes the command to create a UDDI node, with default policies, within a Cloudscape database and datasource. This is a special case only for Cloudscape and creates everything required to run a UDDI node.

  If the Cloudscape database already exists, you will be asked if you want to recreate it. If you choose to recreate the database, your existing database will be deleted and a new one created in its place. If you choose not to recreate the database, the command will exit and a new database will not be created.

  Note: If the application server already accessed the existing Cloudscape database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in "Removing a UDDI registry node" on page 503, restart the server and run the uddiDeploy.jacl script again.

For example, to create a UDDI node called 'MyNode' on server 'server1' on a Windows system, you might enter the following (this assumes server1 is started):

```
wsadmin -wsadmin_classpath C:\Progra~1\IBM\WebSphere\AppServer\derby\lib -f uddiDeploy.jacl
MyNode server1 default
```

If the server is not started the command is:

```
wsadmin -conntype none -wsadmin_classpath C:\Progra~1\IBM\WebSphere\AppServer\derby\lib -f
uddiDeploy.jacl MyNode server1 default
```

(Note that these should be entered as one command on a single line)

2. Linux If you are using DB2, stop the server if it is running. Edit the user profile for the DB2 user that will start the server. Modify the user profile to run the db2profile script which is located in the root directory of the DB2 userid (for example /home/db2inst1/sqllib/db2profile). Alternatively you can run the db2profile script manually by entering the following command, however you will need to do this every time you restart the server :

   ```
   . /home/db2inst1/sqllib/db2profile
   ```

   Note: In the above example, notice that the '.' is followed by a single space character.
3. Click **Applications** → **Enterprise Applications** to display the installed applications. Start the UDDI registry application by selecting the check box next to it and clicking **Start**. Alternatively start the application server if it is not already running; this will automatically start the UDDI registry application. The UDDI node is now active.

   Note: Restarting the UDDI application, or the application server, will always result in the reactivation of the UDDI node, even if the node was previously deactivated.
4. Click **UDDI** → **UDDI Nodes >** *UDDI_node_id* to display the properties page for the UDDI registry node. Set **Prefix for generated discoveryURLs** to a valid URL for your configuration. This property specifies the URL prefix that is applied to generated discovery URLs that are used by the HTTP GET service for UDDI version 2.

Follow the instructions in "Using the UDDI registry Installation Verification Program (IVP)" on page 441 to verify that you have successfully set up the UDDI node.

Setting up a default UDDI node

Use this task to create a UDDI node with predetermined property values. You will not be able to change the mandatory node properties, such as node ID, either during the creation of the node or afterwards.

Such a node is suitable for initial evaluation of the UDDI registry and for development and test purposes. If you want to choose the mandatory node properties yourself, set up a customized node as detailed in “Setting up a customized UDDI node” on page 427.

1. Create a database schema to hold the UDDI registry by completing one of the following tasks, ensuring that you use the default node options where specified:
   - “Creating a DB2 distributed database for the UDDI registry”
   - “Creating a DB2 for z/OS database for the UDDI registry” on page 418
   - “Creating a Cloudscape database for the UDDI registry” on page 421
   - “Creating an Oracle database for the UDDI registry” on page 422
2. Set up a datasource for the UDDI registry application to use to access the database, as described in “Creating a data source for the UDDI registry” on page 423.
3. Deploy the UDDI registry application, as described in “Deploying the UDDI registry application” on page 426.
4. Linux If you are using DB2, stop the server if it is running. Edit the user profile for the DB2 user that will start the server. Modify the user profile to run the db2profile script which is located in the root directory of the DB2 userid (for example /home/db2inst1/sqllib/db2profile). Alternatively you can run the db2profile script manually by entering the following command, however you will need to do this every time you restart the server :

   ```
   . /home/db2inst1/sqllib/db2profile
   ```

   Note: In the above example, notice that the ’.’ is followed by a single space character.
5. Click **Applications** → **Enterprise Applications** to display the installed applications. Start the UDDI registry application by selecting the check box next to it and clicking **Start**. Alternatively start the application server if it is not already running; this will automatically start the UDDI registry application. The UDDI node is now active.

   Note: Restarting the UDDI application, or the application server, will always result in the reactivation of the UDDI node, even if the node was previously deactivated.
6. Click **UDDI** → **UDDI Nodes >** *UDDI_node_id* to display the properties page for the UDDI registry node. Set **Prefix for generated discoveryURLs** to a valid URL for your configuration. This property specifies the URL prefix that is applied to generated discovery URLs that are used by the HTTP GET service for UDDI version 2.

As you have chosen to use a default UDDI node, the node will be initialized when the UDDI application is started for the first time. Follow the instructions in “Using the UDDI registry Installation Verification Program (IVP)” on page 441 to verify that you have successfully set up the UDDI node.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used, and suggested values are:

<DataBaseName>
: is the name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI documentation. If you use some other name, then you should substitute that name whenever you see ’UDDI30’ in other sections of the documentation.

<DB2UserID>
: is a DB2 userid with administrative privileges.

<DB2Password>
: is the password for the DB2 userid.

<BufferPoolName>
is the name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, as the buffer pool is created as part of this task.

<TableSpaceName>
is the name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
is the name of a temporary table space. A suggested value is udditstemp, but any name can be used, as the temporary table space is created as part of this task.

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt (for Windows platforms, enter `db2cmd` and then enter `db2` in the new DB2 window).
3. Run the following command to setup the DB2 environment variables:
   ```
   set DB2CODEPAGE=1208
   ```
4. Create the DB2 database by entering the following command:
   ```
   create database <DataBaseName> using codeset UTF-8 territory en
   ```
   where <DataBaseName> is the name of the database being created.
5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`
   c. `connect reset`
   d. `terminate`
   e. `force application all`
   f. `terminate`
   g. `stop`
   h. `start`
7. Create further database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:

a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
b. `db2 -tvf uddi30crt_20_tables_generic.sql`
c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`
f. `db2 -tvf uddi30crt_40_views_generic.sql`
g. `db2 -tvf uddi30crt_45_views_db2udb.sql`
h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`
i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`

9. Run this step only if you want your database to be used as a default UDDI node. Enter the following command:
   `db2 -tvf uddi30crt_70_insert_default_database_indicator.sql`

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used, and suggested values are:

<DataBaseName>
is the name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI documentation. If you use some other name, then you should substitute that name whenever you see 'UDDI30' in other sections of the documentation.

<DB2UserID>
is a DB2 userid with administrative privileges.

<DB2Password>
is the password for the DB2 userid.

<BufferPoolName>
is the name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, as the buffer pool is created as part of this task.

<TableSpaceName>
is the name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
is the name of a temporary table space. A suggested value is udditstemp, but any name can be used, as the temporary table space is created as part of this task.

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt (for Windows platforms, enter `db2cmd` and then enter `db2` in the new DB2 window).
3. Run the following command to setup the DB2 environment variables:
   `set DB2CODEPAGE=1208`
4. Create the DB2 database by entering the following command:

```
create database <DataBaseName> using codeset UTF-8 territory en
```

where <DataBaseName> is the name of the database being created.

5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`
   c. `connect reset`
   d. `terminate`
   e. `force application all`
   f. `terminate`
   g. `stop`
   h. `start`
7. Create further database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:
   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
   b. `db2 -tvf uddi30crt_20_tables_generic.sql`
   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`
   f. `db2 -tvf uddi30crt_40_views_generic.sql`
   g. `db2 -tvf uddi30crt_45_views_db2udb.sql`
   h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`
   i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`
9. Run this step only if you want your database to be used as a default UDDI node. Enter the following command:
   ```
   db2 -tvf uddi30crt_70_insert_default_database_indicator.sql
   ```

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for z/OS database for the UDDI registry

Perform this task if you want to use DB2 for z/OS as the database store for your UDDI registry data.

In order to connect from a machine running a distributed operating system to a remote DB2 database on the z/OS operating system, you must have DB2 Version 8.2 or later. You must also have a DB2 Connect license (see the DB2 documentation for more information).

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Copy the createddl.sh script supplied in *app_server_root*/UDDIReg/rexx, to a temporary directory of your choice.
2. Using the UNIX System Services (USS) command prompt, edit the copy of the createddl.sh script, as follows:
   a. Search for the text 'Define some constants'.
   b. If you have installed WebSphere Application Server in a non default location, update the root_dir constant to reflect this (note that the UDDIReg directory must remain at the end of the path).
   c. Update the temp_dir constant to a temporary directory of your choice, if you do not want to accept the default.
3. Using the USS command prompt, run the copy of the createddl.sh script by entering the following command:

   ```
   createddl.sh database_name tablespace_name hlq
   ```

   where the parameters are as follows:

   database_name
   This is the name which will be used when defining the required DB2 tables and other components. The default is UDDI30.

   tablespace_name
   This is the tablespace in which the database's tables will be defined. The default is UDDI30TS.

   hlq This is the high level qualifier under which the SQL and JCL partitioned datasets (PDS) will be created. The default is IBMUSER.

   The script generates the partitioned data sets *hlq*.UDDI.SQL and *hlq*.UDDI.JCL, containing members that are required for subsequent steps.Using the default parameters listed above, a successful execution of the script results in the following output:

   ```
   database.tablespace = UDDI30.UDDI30TS
    HLQ = IBMUSER
    (  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_10_prereq_db2.sql
    ( 436) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_20_tables_generic.sql
    ( 136) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_25_tables_db2udb.sql
    ( 452) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_30_constraints_generic.sql
    (  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_35_constraints_db2udb.sql
    ( 559) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_40_views_generic.sql
    (  94) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_45_views_db2udb.sql
    ( 329) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_50_triggers_db2udb.sql
    (  16) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_60_insert_initial_static_
           data.sql
    (  39) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_70_insert_default_database_
           indicator.sql
    Conversion complete
    /tmp/udditmp/makedb71.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB71)
    /tmp/udditmp/makedb81.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB81)
    /tmp/udditmp/table.sql        ===> IBMUSER.UDDI.SQL(TABLE)
    /tmp/udditmp/table7.sql       ===> IBMUSER.UDDI.SQL(TABLE7)
    /tmp/udditmp/index.sql        ===> IBMUSER.UDDI.SQL(INDEX)
    /tmp/udditmp/view.sql         ===> IBMUSER.UDDI.SQL(VIEW)
    /tmp/udditmp/trigger.sql      ===> IBMUSER.UDDI.SQL(TRIGGER)
   ```

```
/tmp/udditmp/alter.sql        ===> IBMUSER.UDDI.SQL(ALTER)
/tmp/udditmp/initial.sql      ===> IBMUSER.UDDI.SQL(INITIAL)
/tmp/udditmp/insert.sql       ===> IBMUSER.UDDI.SQL(INSERT)
```

4. There are two sample jobs in the JCL library for creating the DB2 database, one for DB2 version 7 and one for DB2 version 8. The JCL for these jobs can be found in members MAKEDB71 and MAKEDB81 respectively, in the *hlq*.UDDI.JCL PDS. These JCL scripts are templates; modify the template in the appropriate MAKEDB member according to your DB2 setup and whether you want a default or a customized UDDI node:
   - Add or modify the JOB accounting information, if required.
   - If you used a different high level qualifier from the default when running the script in step one, ensure that all occurrences of IBMUSER are changed to the qualifier that you specified.
   - If you do not want your database to be used as a default UDDI node, comment out the line of the job which specifies the INSERT member of the SQL PDS; this should be the last line in the job.
   - Ensure that all occurrences of the LIB parameter correctly reflect the directory into which you installed DB2.
5. Use TSO to submit the job that you modified in the previous step. The job will create the DB2 database.

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for z/OS database for the UDDI registry

Perform this task if you want to use DB2 for z/OS as the database store for your UDDI registry data.

In order to connect from a machine running a distributed operating system to a remote DB2 database on the z/OS operating system, you must have DB2 Version 8.2 or later. You must also have a DB2 Connect license (see the DB2 documentation for more information).

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Copy the createddl.sh script supplied in *app_server_root*/UDDIReg/rexx, to a temporary directory of your choice.
2. Using the UNIX System Services (USS) command prompt, edit the copy of the createddl.sh script, as follows:
   a. Search for the text 'Define some constants'.
   b. If you have installed WebSphere Application Server in a non default location, update the root_dir constant to reflect this (note that the UDDIReg directory must remain at the end of the path).
   c. Update the temp_dir constant to a temporary directory of your choice, if you do not want to accept the default.
3. Using the USS command prompt, run the copy of the createddl.sh script by entering the following command:

   ```
   createddl.sh database_name tablespace_name hlq
   ```

   where the parameters are as follows:

   database_name
   This is the name which will be used when defining the required DB2 tables and other components. The default is UDDI30.

   tablespace_name
   This is the tablespace in which the database's tables will be defined. The default is UDDI30TS.

hlq This is the high level qualifier under which the SQL and JCL partitioned datasets (PDS) will be created. The default is IBMUSER.

The script generates the partitioned data sets *hlq*.UDDI.SQL and *hlq*.UDDI.JCL, containing members that are required for subsequent steps.Using the default parameters listed above, a successful execution of the script results in the following output:

```
database.tablespace = UDDI30.UDDI30TS
 HLQ = IBMUSER
 (  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_10_prereq_db2.sql
 ( 436) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_20_tables_generic.sql
 ( 136) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_25_tables_db2udb.sql
 ( 452) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_30_constraints_generic.sql
 (  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_35_constraints_db2udb.sql
 ( 559) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_40_views_generic.sql
 (  94) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_45_views_db2udb.sql
 ( 329) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_50_triggers_db2udb.sql
 (  16) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_60_insert_initial_static_
        data.sql
 (  39) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_70_insert_default_database_
        indicator.sql
 Conversion complete
 /tmp/udditmp/makedb71.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB71)
 /tmp/udditmp/makedb81.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB81)
 /tmp/udditmp/table.sql        ===> IBMUSER.UDDI.SQL(TABLE)
 /tmp/udditmp/table7.sql       ===> IBMUSER.UDDI.SQL(TABLE7)
 /tmp/udditmp/index.sql        ===> IBMUSER.UDDI.SQL(INDEX)
 /tmp/udditmp/view.sql         ===> IBMUSER.UDDI.SQL(VIEW)
 /tmp/udditmp/trigger.sql      ===> IBMUSER.UDDI.SQL(TRIGGER)
 /tmp/udditmp/alter.sql        ===> IBMUSER.UDDI.SQL(ALTER)
 /tmp/udditmp/initial.sql      ===> IBMUSER.UDDI.SQL(INITIAL)
 /tmp/udditmp/insert.sql       ===> IBMUSER.UDDI.SQL(INSERT)
```

4. There are two sample jobs in the JCL library for creating the DB2 database, one for DB2 version 7 and one for DB2 version 8. The JCL for these jobs can be found in members MAKEDB71 and MAKEDB81 respectively, in the *hlq*.UDDI.JCL PDS. These JCL scripts are templates; modify the template in the appropriate MAKEDB member according to your DB2 setup and whether you want a default or a customized UDDI node:
   - Add or modify the JOB accounting information, if required.
   - If you used a different high level qualifier from the default when running the script in step one, ensure that all occurrences of IBMUSER are changed to the qualifier that you specified.
   - If you do not want your database to be used as a default UDDI node, comment out the line of the job which specifies the INSERT member of the SQL PDS; this should be the last line in the job.
   - Ensure that all occurrences of the LIB parameter correctly reflect the directory into which you installed DB2.
5. Use TSO to submit the job that you modified in the previous step. The job will create the DB2 database.

Continue with setting up and deploying your UDDI registry node.

Creating a Cloudscape database for the UDDI registry

Perform this task if you want to use Cloudscape (embedded or network) as the database store (either local or remote) for your UDDI registry. You need only perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

The commands below use a number of arguments for which you need to enter appropriate values. You should decide the values that you will use before you start. The arguments used, and suggested values, are:

arg1 is the path of the SQL files, which on a standard installation will be *app_server_root*/UDDIReg/databasescripts

arg2 is the path to the location where you want to install the Cloudscape database.

For example, *app_server_root*/profiles/*profile_name*/databases/com.ibm.uddi

arg3 is the name of the Cloudscape database. A recommended value is UDDI30, and this name is assumed throughout the UDDI documentation. If you use another name, you should substitute that name whenever you see 'UDDI30' in other sections of the UDDI documentation.

arg4 is an optional argument, and must either be omitted or be the string 'DEFAULT'. DEFAULT should only be specified if you want your database to be used as a default UDDI node. Note that this argument is case sensitive.

1. Run the following Java -jar command from the *app_server_root*/UDDIReg/databaseScripts directory, to create a UDDI Cloudscape database using UDDIDerbyCreate.jar. Windows

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib;app_server_root/java/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   AIX Linux HP-UX Solaris

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib:app_server_root/java/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   If the Cloudscape database already exists, you will be asked if you want to recreate it. If you choose to recreate the database, your existing database will be deleted and a new one created in its place. If you choose not to recreate the database, the command will exit and a new database will not be created.

   Note: If the application server already accessed the existing Cloudscape database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in "Removing a UDDI registry node" on page 503, restart the server and run the uddiDeploy.jacl script again.

2. If you are using a remote database (which requires network Cloudscape), or you want to use network Cloudscape for other reasons, for example if you want to use Cloudscape with a cluster, configure the Cloudscape Network Server framework as described in the managing derby network server section of the Cloudscape information center.

Continue with setting up and deploying your UDDI registry node.

Creating an Oracle database for the UDDI registry

Note: Only Version 9i[1] and Oracle 10g[2]

1.

Restrictions:

discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters; instanceParms (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 8192 characters; overviewURL (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 4096 characters; Digital Signature maximum 4000 bytes.

2.

Restrictions:

discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters.

Perform this task if you want to use Oracle as the database store (either local or remote) for your UDDI registry data. You need only do this once for each UDDI registry, as part of Setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

This task creates three new schemas: ibmuddi, ibmudi30 and ibmuds30. You will be unable to complete this task if you already have existing schemas with these names.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used are:

<OracleUserID>
is the Oracle userid to be used to create the database.

<OraclePassword>
is the password for the Oracle userid.

1. Run the following commands:
   a. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_10_prereq_oracle.sql`
   b. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_20_tables_generic.sql`
   c. Complete one of the following actions depending on your level of Oracle:
      - For Oracle 9i:
        `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle_pre10g.sql`
      - For Version 10g and later:
        `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle.sql`
   d. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_30_constraints_generic.sql`
   e. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_35_constraints_oracle.sql`
   f. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_40_views_generic.sql`
   g. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_45_views_oracle.sql`
   h. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_50_triggers_oracle.sql`
   i. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_60_insert_initial_static_data.sql`
2. This last command should only be run if you want the database to be used as a default UDDI node.
   `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_70_insert_default_database_indicator.sql`

Continue with setting up and deploying your UDDI registry node.

Creating a data source for the UDDI registry

You must have already created the database for the UDDI registry.

Note: If you are connecting to a remote DB2 database on the z/OS operating system, you must have installed a DB2 Connect license (See the DB2 documentation for more information).

Perform this task as part of setting up and deploying a new UDDI registry. The data source is used by the UDDI registry to access the UDDI database.

1. Create a J2C Authentication Data Entry (not required for embedded Cloudscape, but required for network Cloudscape):

a. Click **Security** → **Secure administration, applications, and infrastructire** → **[Authentication] Java Authentication and Authorization Service** → **J2C authentication data**.

b. Click **New** to create a new J2C authentication data entry

c. Fill in the details as follows:

Alias a suitable (short) name, such as ″UDDIAlias″

Userid

the database userid (such as db2admin for DB2 or IBMUDDI for Oracle), which is used to read and write to the UDDI registry database. For network Cloudscape the userid can be any value.

If you are using a remote DB2 database on the z/OS operating system, the userid must be one that is valid on the remote system.

Password

the password associated with the userid specified above. For network Cloudscape the password can be any value.

Description

a suitable description to describe the chosen userid.

Click **Apply** and then Save the changes to the master configuration.

2. Create a JDBC Provider (if a suitable one does not already exist), using the following table to determine the provider type and implementation type for your chosen database:

Database	Provider type	Implementation type
DB2	DB2 Universal JDBC Driver Provider	Connection Pool data source
Oracle	Oracle JDBC Driver	Connection Pool data source
Embedded Cloudscape	Derby JDBC Driver	Connection Pool data source
Network Cloudscape	Derby Network Server JDBC Driver provider	Connection Pool data source

For details on how to create a JDBC provider, see Creating and configuring a JDBC provider using the administrative console.

3. Create the data source for the UDDI registry by following these steps:

a. Click **Resources** → **JDBC** → **JDBC Providers**.

b. Select the desired 'scope' of the JDBC provider you selected or created earlier. For example, select:

```
Server: yourservername
```

to show the JDBC providers at the server level.

c. Select the JDBC provider created earlier.

d. Under **Additional Properties**, select **Data sources** (not the **Data sources (WebSphere Application Server V4)** option).

e. Click **New** to create a new data source.

f. In the **Create a data source** wizard, enter the following data:

Name a suitable name, such as UDDI Datasource

JNDI name

set to **datasources/uddids** - this value is obligatory.

Note: You must not have any other data sources using this JNDI name. If you have another data source using this JNDI name, then you must either remove it or change its JNDI name. For example, if you have previously created a default UDDI node

using Cloudscape, you should use the uddiRemove.jacl script with the default option to remove the data source and the UDDI application instance, before continuing.

Component-managed authentication alias

- for DB2, Oracle or network Cloudscape, select the alias that you created in step 2 from the pulldown. It will have the node name appended in front of it, for example MyNode/UDDIAlias.
- for embedded Cloudscape leave this set to (none).

g. Click **Next**.

h. On the database specific properties page of the wizard, enter the following data:

- for DB2:

  Database name
  for example:
  `UDDI30`

  Notes:

  – If you are using a remote database on a distributed system, the database name is the alias that you created to reference the database. See Creating a DB2 distributed database.

  – If you are using a remote DB2 database on the z/OS operating system, the database name is the local LOCATION value. To find this value, enter the operator command `-DIS DDF` at the console (or ask your DB2 administrator for the information). Note that this value is case sensitive.

  Driver type
  this applies only if you are using a remote DB2 database on the z/OS operating system. Set this value to 4.

  Server name
  this applies only if you are using a remote DB2 database on the z/OS operating system. Set this value to the IP address of the remote machine that is hosting the database. Use the `-DIS DDF` operator command to find this information (or ask your DB2 administrator for the information).

  Port number
  this applies only if you are using a remote DB2 database on the z/OS operating system. Set this value to the port that the DB2 database is listening on. Use the `-DIS DDF` operator command to find this information (or ask your DB2 administrator for the information).

- for Oracle - **URL** - for example:

  ```
  jdbc:oracle:oci8:@<Oracle database name>
  ```

  This applies to local and remote Oracle databases.

- for Cloudscape (embedded or network) - **Database name** - for example:

  app_server_root/profiles/*profile_name*/databases/com.ibm.uddi/UDDI30

  For network Cloudscape, also make sure that the **Server name** and **Port number** match the network server.

  Leave all other fields unchanged.

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

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

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

Description
a suitable description

Category
set to uddi

Data store helper class name
filled in for you as:

Database	Data store helper class name
DB2	com.ibm.websphere.rsadapter.DB2DataStoreHelper, or com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper if you are using a remote DB2 database on the z/OS operating system
Oracle 9i	com.ibm.websphere.rsadapter.OracleDataStoreHelper
Oracle 10g	com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper
Embedded Cloudscape	com.ibm.websphere.rsadapter.DerbyDataStoreHelper
Network Cloudscape	com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper

Mapping-configuration alias
set to DefaultPrincipalMapping

k. Click **Apply** and save the changes to the master configuration.

4. Test the connection to your UDDI database by selecting the check box next to the data source and clicking **Test connection**. You will see a message similar to ″Test Connection for datasource UDDI Datasource on server server1 at node MyNode was successful″. If you do not see this message investigate the problem with the help of the error message.

Continue with setting up and deploying your UDDI registry node.

Deploying the UDDI registry application

You must have already created the database and datasource for the UDDI registry.

Use this task as part of “Setting up a default UDDI node” on page 414 or “Setting up a customized UDDI node” on page 427.

Run the uddiDeploy.jacl script as shown below, from the *app_server_root*/bin directory. This script deploys the UDDI registry to a server specified by you.

Linux

Note: If you are using either the UNIX or Linux operating systems, add the `.sh` suffix to the wsadmin command.

```
wsadmin [-conntype none] [-profileName profile_name] -f uddiDeploy.jacl
                node_name
                server_name
```

where

- ’-profileName *profile_name*’ is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
- ’-conntype none’ is optional, and is only needed if the application server is not running.
- *node_name* is the name of the WebSphere node on which the target server runs. Note that the node name is case sensitive.
- *server_name* is the name of the target server on which you wish to deploy the UDDI registry, such as server1. Note the server name entered is case sensitive.

For example, to deploy UDDI on node 'MyNode' and server 'server1' on a Windows system, (assuming that server1 is already started):

```
wsadmin -f uddiDeploy.jacl MyNode server1
```

You can also deploy the UDDI application (the uddi.ear file) using the administrative console, in the normal way. However, some steps that are performed automatically by the uddiDeploy.jacl script do not take place in this scenario. If you use the administrative console to install the UDDI application, you must perform some actions manually, according to the following steps:

1. Install the application.
2. Click **Applications** → **Enterprise Applications** > *uddi_application* > **[Detail Properties] Class loading and update detection**.
3. Ensure that **Class loader order** is set to **Classes loaded with application class loader first**.
4. Ensure that **WAR class loader policy** is set to **Single class loader for application**.

Continue with setting up the UDDI node.

Setting up a customized UDDI node

Use this task to set up a UDDI node with property values that are chosen by you. You will not be able to change the mandatory node properties, such as node ID, after the initialization of the node. Such a node is suitable for production purposes.

1. Review the information in "Database considerations for production use of the UDDI registry" on page 413 to help you decide which database system to use, then create a database schema to hold the UDDI registry by completing one of the following tasks, ensuring that you do NOT use the default node options where specified:
   - "Creating a DB2 distributed database for the UDDI registry" on page 415
   - "Creating a DB2 for z/OS database for the UDDI registry" on page 418
   - "Creating a Cloudscape database for the UDDI registry" on page 421
   - "Creating an Oracle database for the UDDI registry" on page 422
2. Set up a datasource for the UDDI registry application to use to access the database, as described in "Creating a data source for the UDDI registry" on page 423.
3. Deploy the UDDI registry application, as described in "Deploying the UDDI registry application" on page 426.
4. Linux If you are using DB2, stop the server if it is running. Edit the user profile for the DB2 user that will start the server. Modify the user profile to run the db2profile script which is located in the root directory of the DB2 userid (for example /home/db2inst1/sqllib/db2profile). Alternatively you can run the db2profile script manually by entering the following command, however you will need to do this every time you restart the server :

   ```
   . /home/db2inst1/sqllib/db2profile
   ```

   Note: In the above example, notice that the '.' is followed by a single space character.
5. Click **Applications** → **Enterprise Applications** to display the installed applications. Start the UDDI registry application by selecting the check box next to it and clicking **Start**. Alternatively start the application server if it is not already running; this will automatically start the UDDI registry application. The UDDI node is now active.

   Note: Restarting the UDDI application, or the application server, will always result in the reactivation of the UDDI node, even if the node was previously deactivated.

As you have chosen a user customized UDDI node, you will need to set the properties for the UDDI node using UDDI administration, and initialize the node before it is ready to accept UDDI requests. See "Initializing the UDDI registry node" on page 439 for details.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used, and suggested values are:

<DataBaseName>
is the name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI documentation. If you use some other name, then you should substitute that name whenever you see 'UDDI30' in other sections of the documentation.

<DB2UserID>
is a DB2 userid with administrative privileges.

<DB2Password>
is the password for the DB2 userid.

<BufferPoolName>
is the name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, as the buffer pool is created as part of this task.

<TableSpaceName>
is the name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
is the name of a temporary table space. A suggested value is udditstemp, but any name can be used, as the temporary table space is created as part of this task.

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt (for Windows platforms, enter `db2cmd` and then enter `db2` in the new DB2 window).
3. Run the following command to setup the DB2 environment variables:
   ```
   set DB2CODEPAGE=1208
   ```
4. Create the DB2 database by entering the following command:
   ```
   create database <DataBaseName> using codeset UTF-8 territory en
   ```
   where <DataBaseName> is the name of the database being created.
5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`
   c. `connect reset`
   d. `terminate`

e. `force application all`

f. `terminate`

g. `stop`

h. `start`

7. Create further database structures by entering the following commands:

   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`

   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`

   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`

8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:

   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`

   b. `db2 -tvf uddi30crt_20_tables_generic.sql`

   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`

   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`

   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`

   f. `db2 -tvf uddi30crt_40_views_generic.sql`

   g. `db2 -tvf uddi30crt_45_views_db2udb.sql`

   h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`

   i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`

9. Run this step only if you want your database to be used as a default UDDI node. Enter the following command:

   `db2 -tvf uddi30crt_70_insert_default_database_indicator.sql`

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used, and suggested values are:

<DataBaseName>
is the name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI documentation. If you use some other name, then you should substitute that name whenever you see 'UDDI30' in other sections of the documentation.

<DB2UserID>
is a DB2 userid with administrative privileges.

<DB2Password>
is the password for the DB2 userid.

<BufferPoolName>
is the name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, as the buffer pool is created as part of this task.

<TableSpaceName>
is the name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
is the name of a temporary table space. A suggested value is udditstemp, but any name can be used, as the temporary table space is created as part of this task.

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt (for Windows platforms, enter `db2cmd` and then enter `db2` in the new DB2 window).
3. Run the following command to setup the DB2 environment variables:
   ```
   set DB2CODEPAGE=1208
   ```
4. Create the DB2 database by entering the following command:
   ```
   create database <DataBaseName> using codeset UTF-8 territory en
   ```
   where <DataBaseName> is the name of the database being created.
5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`
   c. `connect reset`
   d. `terminate`
   e. `force application all`
   f. `terminate`
   g. `stop`
   h. `start`
7. Create further database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:
   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
   b. `db2 -tvf uddi30crt_20_tables_generic.sql`
   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`

f. `db2 -tvf uddi30crt_40_views_generic.sql`

g. `db2 -tvf uddi30crt_45_views_db2udb.sql`

h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`

i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`

9. Run this step only if you want your database to be used as a default UDDI node. Enter the following command:

   ```
   db2 -tvf uddi30crt_70_insert_default_database_indicator.sql
   ```

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for z/OS database for the UDDI registry

Perform this task if you want to use DB2 for z/OS as the database store for your UDDI registry data.

In order to connect from a machine running a distributed operating system to a remote DB2 database on the z/OS operating system, you must have DB2 Version 8.2 or later. You must also have a DB2 Connect license (see the DB2 documentation for more information).

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Copy the createddl.sh script supplied in *app_server_root*/UDDIReg/rexx, to a temporary directory of your choice.
2. Using the UNIX System Services (USS) command prompt, edit the copy of the createddl.sh script, as follows:
   a. Search for the text 'Define some constants'.
   b. If you have installed WebSphere Application Server in a non default location, update the root_dir constant to reflect this (note that the UDDIReg directory must remain at the end of the path).
   c. Update the temp_dir constant to a temporary directory of your choice, if you do not want to accept the default.
3. Using the USS command prompt, run the copy of the createddl.sh script by entering the following command:

   ```
   createddl.sh database_name tablespace_name hlq
   ```

   where the parameters are as follows:

   database_name
   This is the name which will be used when defining the required DB2 tables and other components. The default is UDDI30.

   tablespace_name
   This is the tablespace in which the database's tables will be defined. The default is UDDI30TS.

   hlq This is the high level qualifier under which the SQL and JCL partitioned datasets (PDS) will be created. The default is IBMUSER.

   The script generates the partitioned data sets *hlq*.UDDI.SQL and *hlq*.UDDI.JCL, containing members that are required for subsequent steps.Using the default parameters listed above, a successful execution of the script results in the following output:

   ```
   database.tablespace = UDDI30.UDDI30TS
    HLQ = IBMUSER
    (  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_10_prereq_db2.sql
    ( 436) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_20_tables_generic.sql
    ( 136) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_25_tables_db2udb.sql
    ( 452) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_30_constraints_generic.sql
   ```

```
(  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_35_constraints_db2udb.sql
( 559) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_40_views_generic.sql
(  94) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_45_views_db2udb.sql
( 329) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_50_triggers_db2udb.sql
(  16) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_60_insert_initial_static_
       data.sql
(  39) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_70_insert_default_database_
       indicator.sql
Conversion complete
/tmp/udditmp/makedb71.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB71)
/tmp/udditmp/makedb81.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB81)
/tmp/udditmp/table.sql        ===> IBMUSER.UDDI.SQL(TABLE)
/tmp/udditmp/table7.sql       ===> IBMUSER.UDDI.SQL(TABLE7)
/tmp/udditmp/index.sql        ===> IBMUSER.UDDI.SQL(INDEX)
/tmp/udditmp/view.sql         ===> IBMUSER.UDDI.SQL(VIEW)
/tmp/udditmp/trigger.sql      ===> IBMUSER.UDDI.SQL(TRIGGER)
/tmp/udditmp/alter.sql        ===> IBMUSER.UDDI.SQL(ALTER)
/tmp/udditmp/initial.sql      ===> IBMUSER.UDDI.SQL(INITIAL)
/tmp/udditmp/insert.sql       ===> IBMUSER.UDDI.SQL(INSERT)
```

4. There are two sample jobs in the JCL library for creating the DB2 database, one for DB2 version 7 and one for DB2 version 8. The JCL for these jobs can be found in members MAKEDB71 and MAKEDB81 respectively, in the *hlq*.UDDI.JCL PDS. These JCL scripts are templates; modify the template in the appropriate MAKEDB member according to your DB2 setup and whether you want a default or a customized UDDI node:
   - Add or modify the JOB accounting information, if required.
   - If you used a different high level qualifier from the default when running the script in step one, ensure that all occurrences of IBMUSER are changed to the qualifier that you specified.
   - If you do not want your database to be used as a default UDDI node, comment out the line of the job which specifies the INSERT member of the SQL PDS; this should be the last line in the job.
   - Ensure that all occurrences of the LIB parameter correctly reflect the directory into which you installed DB2.
5. Use TSO to submit the job that you modified in the previous step. The job will create the DB2 database.

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for z/OS database for the UDDI registry

Perform this task if you want to use DB2 for z/OS as the database store for your UDDI registry data.

In order to connect from a machine running a distributed operating system to a remote DB2 database on the z/OS operating system, you must have DB2 Version 8.2 or later. You must also have a DB2 Connect license (see the DB2 documentation for more information).

You only need to perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

1. Copy the createddl.sh script supplied in *app_server_root*/UDDIReg/rexx, to a temporary directory of your choice.
2. Using the UNIX System Services (USS) command prompt, edit the copy of the createddl.sh script, as follows:
   a. Search for the text 'Define some constants'.
   b. If you have installed WebSphere Application Server in a non default location, update the root_dir constant to reflect this (note that the UDDIReg directory must remain at the end of the path).

c. Update the temp_dir constant to a temporary directory of your choice, if you do not want to accept the default.

3. Using the USS command prompt, run the copy of the createddl.sh script by entering the following command:

```
createddl.sh database_name tablespace_name hlq
```

where the parameters are as follows:

database_name
This is the name which will be used when defining the required DB2 tables and other components. The default is UDDI30.

tablespace_name
This is the tablespace in which the database's tables will be defined. The default is UDDI30TS.

hlq This is the high level qualifier under which the SQL and JCL partitioned datasets (PDS) will be created. The default is IBMUSER.

The script generates the partitioned data sets *hlq*.UDDI.SQL and *hlq*.UDDI.JCL, containing members that are required for subsequent steps.Using the default parameters listed above, a successful execution of the script results in the following output:

```
database.tablespace = UDDI30.UDDI30TS
 HLQ = IBMUSER
 (  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_10_prereq_db2.sql
 ( 436) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_20_tables_generic.sql
 ( 136) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_25_tables_db2udb.sql
 ( 452) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_30_constraints_generic.sql
 (  14) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_35_constraints_db2udb.sql
 ( 559) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_40_views_generic.sql
 (  94) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_45_views_db2udb.sql
 ( 329) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_50_triggers_db2udb.sql
 (  16) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_60_insert_initial_static_
        data.sql
 (  39) /WebSphere/V6R0M0/AppServer/UDDIReg/databaseScripts/uddi30crt_70_insert_default_database_
        indicator.sql
 Conversion complete
 /tmp/udditmp/makedb71.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB71)
 /tmp/udditmp/makedb81.jcl     ===> IBMUSER.UDDI.JCL(MAKEDB81)
 /tmp/udditmp/table.sql        ===> IBMUSER.UDDI.SQL(TABLE)
 /tmp/udditmp/table7.sql       ===> IBMUSER.UDDI.SQL(TABLE7)
 /tmp/udditmp/index.sql        ===> IBMUSER.UDDI.SQL(INDEX)
 /tmp/udditmp/view.sql         ===> IBMUSER.UDDI.SQL(VIEW)
 /tmp/udditmp/trigger.sql      ===> IBMUSER.UDDI.SQL(TRIGGER)
 /tmp/udditmp/alter.sql        ===> IBMUSER.UDDI.SQL(ALTER)
 /tmp/udditmp/initial.sql      ===> IBMUSER.UDDI.SQL(INITIAL)
 /tmp/udditmp/insert.sql       ===> IBMUSER.UDDI.SQL(INSERT)
```

4. There are two sample jobs in the JCL library for creating the DB2 database, one for DB2 version 7 and one for DB2 version 8. The JCL for these jobs can be found in members MAKEDB71 and MAKEDB81 respectively, in the *hlq*.UDDI.JCL PDS. These JCL scripts are templates; modify the template in the appropriate MAKEDB member according to your DB2 setup and whether you want a default or a customized UDDI node:
   - Add or modify the JOB accounting information, if required.
   - If you used a different high level qualifier from the default when running the script in step one, ensure that all occurrences of IBMUSER are changed to the qualifier that you specified.
   - If you do not want your database to be used as a default UDDI node, comment out the line of the job which specifies the INSERT member of the SQL PDS; this should be the last line in the job.
   - Ensure that all occurrences of the LIB parameter correctly reflect the directory into which you installed DB2.
5. Use TSO to submit the job that you modified in the previous step. The job will create the DB2 database.

Continue with setting up and deploying your UDDI registry node.

Creating a Cloudscape database for the UDDI registry

Perform this task if you want to use Cloudscape (embedded or network) as the database store (either local or remote) for your UDDI registry. You need only perform this task once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

The commands below use a number of arguments for which you need to enter appropriate values. You should decide the values that you will use before you start. The arguments used, and suggested values, are:

arg1 is the path of the SQL files, which on a standard installation will be *app_server_root*/UDDIReg/databasescripts

arg2 is the path to the location where you want to install the Cloudscape database.

For example, *app_server_root*/profiles/*profile_name*/databases/com.ibm.uddi

arg3 is the name of the Cloudscape database. A recommended value is UDDI30, and this name is assumed throughout the UDDI documentation. If you use another name, you should substitute that name whenever you see 'UDDI30' in other sections of the UDDI documentation.

arg4 is an optional argument, and must either be omitted or be the string 'DEFAULT'. DEFAULT should only be specified if you want your database to be used as a default UDDI node. Note that this argument is case sensitive.

1. Run the following Java -jar command from the *app_server_root*/UDDIReg/databaseScripts directory, to create a UDDI Cloudscape database using UDDIDerbyCreate.jar. Windows

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib;app_server_root/java/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   AIX Linux HP-UX Solaris

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib:app_server_root/java/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   If the Cloudscape database already exists, you will be asked if you want to recreate it. If you choose to recreate the database, your existing database will be deleted and a new one created in its place. If you choose not to recreate the database, the command will exit and a new database will not be created.

   Note: If the application server already accessed the existing Cloudscape database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in "Removing a UDDI registry node" on page 503, restart the server and run the uddiDeploy.jacl script again.

2. If you are using a remote database (which requires network Cloudscape), or you want to use network Cloudscape for other reasons, for example if you want to use Cloudscape with a cluster, configure the Cloudscape Network Server framework as described in the managing derby network server section of the Cloudscape information center.

Continue with setting up and deploying your UDDI registry node.

Creating an Oracle database for the UDDI registry

Note: Only Version 9i[3] and Oracle 10g[4]

Perform this task if you want to use Oracle as the database store (either local or remote) for your UDDI registry data. You need only do this once for each UDDI registry, as part of Setting up and deploying a UDDI registry.

If you are creating a remote database, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

This task creates three new schemas: ibmuddi, ibmudi30 and ibmuds30. You will be unable to complete this task if you already have existing schemas with these names.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used are:

<OracleUserID>
is the Oracle userid to be used to create the database.

<OraclePassword>
is the password for the Oracle userid.

1. Run the following commands:
   a. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_10_prereq_oracle.sql`
   b. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_20_tables_generic.sql`
   c. Complete one of the following actions depending on your level of Oracle:
      - For Oracle 9i:
        `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle_pre10g.sql`
      - For Version 10g and later:
        `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle.sql`
   d. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_30_constraints_generic.sql`
   e. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_35_constraints_oracle.sql`
   f. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_40_views_generic.sql`
   g. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_45_views_oracle.sql`
   h. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_50_triggers_oracle.sql`
   i. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_60_insert_initial_static_data.sql`

3.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters; instanceParms (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 8192 characters; overviewURL (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 4096 characters; Digital Signature maximum 4000 bytes.

4.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters.

2. This last command should only be run if you want the database to be used as a default UDDI node.
   ```
   sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_70_insert_default_database_indicator.sql
   ```

Continue with setting up and deploying your UDDI registry node.

Creating a data source for the UDDI registry

You must have already created the database for the UDDI registry.

Note: If you are connecting to a remote DB2 database on the z/OS operating system, you must have installed a DB2 Connect license (See the DB2 documentation for more information).

Perform this task as part of setting up and deploying a new UDDI registry. The data source is used by the UDDI registry to access the UDDI database.

1. Create a J2C Authentication Data Entry (not required for embedded Cloudscape, but required for network Cloudscape):
   a. Click **Security** → **Secure administration, applications, and infrastructire** → **[Authentication] Java Authentication and Authorization Service** → **J2C authentication data**.
   b. Click **New** to create a new J2C authentication data entry
   c. Fill in the details as follows:

      Alias a suitable (short) name, such as ″UDDIAlias″

      Userid

      the database userid (such as db2admin for DB2 or IBMUDDI for Oracle), which is used to read and write to the UDDI registry database. For network Cloudscape the userid can be any value.

      If you are using a remote DB2 database on the z/OS operating system, the userid must be one that is valid on the remote system.

      Password
      the password associated with the userid specified above. For network Cloudscape the password can be any value.

      Description
      a suitable description to describe the chosen userid.

      Click **Apply** and then Save the changes to the master configuration.
2. Create a JDBC Provider (if a suitable one does not already exist), using the following table to determine the provider type and implementation type for your chosen database:

Database	Provider type	Implementation type
DB2	DB2 Universal JDBC Driver Provider	Connection Pool data source
Oracle	Oracle JDBC Driver	Connection Pool data source
Embedded Cloudscape	Derby JDBC Driver	Connection Pool data source
Network Cloudscape	Derby Network Server JDBC Driver provider	Connection Pool data source

   For details on how to create a JDBC provider, see Creating and configuring a JDBC provider using the administrative console.
3. Create the data source for the UDDI registry by following these steps:
   a. Click **Resources** → **JDBC** → **JDBC Providers**.
   b. Select the desired 'scope' of the JDBC provider you selected or created earlier. For example, select:
      ```
      Server: yourservername
      ```

to show the JDBC providers at the server level.

c. Select the JDBC provider created earlier.

d. Under **Additional Properties**, select **Data sources** (not the **Data sources (WebSphere Application Server V4)** option).

e. Click **New** to create a new data source.

f. In the **Create a data source** wizard, enter the following data:

Name a suitable name, such as UDDI Datasource

JNDI name
set to **datasources/uddids** - this value is obligatory.

Note: You must not have any other data sources using this JNDI name. If you have another data source using this JNDI name, then you must either remove it or change its JNDI name. For example, if you have previously created a default UDDI node using Cloudscape, you should use the uddiRemove.jacl script with the default option to remove the data source and the UDDI application instance, before continuing.

Component-managed authentication alias

- for DB2, Oracle or network Cloudscape, select the alias that you created in step 2 from the pulldown. It will have the node name appended in front of it, for example MyNode/UDDIAlias.
- for embedded Cloudscape leave this set to (none).

g. Click **Next**.

h. On the database specific properties page of the wizard, enter the following data:

- for DB2:

Database name
for example:
`UDDI30`

Notes:

– If you are using a remote database on a distributed system, the database name is the alias that you created to reference the database. See Creating a DB2 distributed database.

– If you are using a remote DB2 database on the z/OS operating system, the database name is the local LOCATION value. To find this value, enter the operator command `-DIS DDF` at the console (or ask your DB2 administrator for the information). Note that this value is case sensitive.

Driver type
this applies only if you are using a remote DB2 database on the z/OS operating system. Set this value to 4.

Server name
this applies only if you are using a remote DB2 database on the z/OS operating system. Set this value to the IP address of the remote machine that is hosting the database. Use the `-DIS DDF` operator command to find this information (or ask your DB2 administrator for the information).

Port number
this applies only if you are using a remote DB2 database on the z/OS operating system. Set this value to the port that the DB2 database is listening on. Use the `-DIS DDF` operator command to find this information (or ask your DB2 administrator for the information).

- for Oracle - **URL** - for example:

jdbc:oracle:oci8:@<Oracle database name>

This applies to local and remote Oracle databases.

- for Cloudscape (embedded or network) - **Database name** - for example:

  app_server_root/profiles/*profile_name*/databases/com.ibm.uddi/UDDI30

  For network Cloudscape, also make sure that the **Server name** and **Port number** match the network server.

  Leave all other fields unchanged.

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

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

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

Description
a suitable description

Category
set to uddi

Data store helper class name
filled in for you as:

Database	Data store helper class name
DB2	com.ibm.websphere.rsadapter.DB2DataStoreHelper, or com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper if you are using a remote DB2 database on the z/OS operating system
Oracle 9i	com.ibm.websphere.rsadapter.OracleDataStoreHelper
Oracle 10g	com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper
Embedded Cloudscape	com.ibm.websphere.rsadapter.DerbyDataStoreHelper
Network Cloudscape	com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper

Mapping-configuration alias
set to DefaultPrincipalMapping

k. Click **Apply** and save the changes to the master configuration.

4. Test the connection to your UDDI database by selecting the check box next to the data source and clicking **Test connection**. You will see a message similar to ″Test Connection for datasource UDDI Datasource on server server1 at node MyNode was successful″. If you do not see this message investigate the problem with the help of the error message.

Continue with setting up and deploying your UDDI registry node.

Deploying the UDDI registry application

You must have already created the database and datasource for the UDDI registry.

Use this task as part of “Setting up a default UDDI node” on page 414 or “Setting up a customized UDDI node” on page 427.

Run the uddiDeploy.jacl script as shown below, from the *app_server_root*/bin directory. This script deploys the UDDI registry to a server specified by you.

Linux

Note: If you are using either the UNIX or Linux operating systems, add the `.sh` suffix to the wsadmin command.

```
wsadmin [-conntype none] [-profileName profile_name] -f uddiDeploy.jacl
               node_name
               server_name
```

where

- '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
- '-conntype none' is optional, and is only needed if the application server is not running.
- *node_name* is the name of the WebSphere node on which the target server runs. Note that the node name is case sensitive.
- *server_name* is the name of the target server on which you wish to deploy the UDDI registry, such as server1. Note the server name entered is case sensitive.

For example, to deploy UDDI on node 'MyNode' and server 'server1' on a Windows system, (assuming that server1 is already started):

```
wsadmin -f uddiDeploy.jacl MyNode server1
```

You can also deploy the UDDI application (the uddi.ear file) using the administrative console, in the normal way. However, some steps that are performed automatically by the uddiDeploy.jacl script do not take place in this scenario. If you use the administrative console to install the UDDI application, you must perform some actions manually, according to the following steps:

1. Install the application.
2. Click **Applications** → **Enterprise Applications** > *uddi_application* > **[Detail Properties] Class loading and update detection**.
3. Ensure that **Class loader order** is set to **Classes loaded with application class loader first**.
4. Ensure that **WAR class loader policy** is set to **Single class loader for application**.

Continue with setting up the UDDI node.

Initializing the UDDI registry node

Use this topic to initialize a UDDI registry node after set up or migration.

You must have already set up a UDDI registry node, either as a new node or to use for migrating a UDDI registry Version 2 node.

The UDDI registry node has various properties, some of which must be set before initializing the node. There are two categories of UDDI registry node properties:

- Mandatory node properties. These properties must be set before the UDDI node can be initialized. You may set these properties as many times as you wish before initialization. However, once the UDDI node has been initialized, these properties will become read only for the lifetime of that UDDI node. It is very important to set these properties correctly.
- All other properties. These properties may be set before, and after, initialization.

Configure these properties and initialize the node using the UDDI administrative console or JMX management interface.

1. Click **UDDI** → **UDDI Nodes** > *UDDI_node_id* to display the properties page for the UDDI registry node.
2. Set the mandatory node properties to suitable, and valid, values. These properties are indicated by the presence of a '*' next to the input field. The properties are listed below; more information on each property is given in the context help of the administrative console.

UDDI node ID
This must be a text string beginning with 'uddi:' that is unique to this UDDI node. The default value may be sufficient, but if you accept it you should ensure that it is unique.

UDDI node description
This is a text string describing the node.

Root key generator
This must be a text string beginning with 'uddi:' that is unique to this UDDI node. The default value may be sufficient but may contain text, such as 'keyspace_id', that you should modify to match your system. If you accept the default value, ensure that it is unique for this UDDI node.

Prefix for generated discoveryURLs
This should be a valid URL.

3. If you are migrating from Version 2 of the UDDI registry, use the table below to perform the following steps:
   - Set any properties from uddi.properties that **must** remain the same as Version 2.
   - Set any properties from uddi.properties that you would like to keep the same (such as dbMaxResultCount).

Version 2 UDDI property (set in uddi.property file)	Version 3 UDDI Property (set via Administrative Console or UDDI Administrative Interface)	Recommended Version 3 UDDI property setting
dbMaxResultCount	maximum inquiry response set size	You might want to retain the value from Version 2, but can safely change this (or use the default)
persister	no equivalent	Not applicable
defaultLanguage	default language code	You are recommended to retain the value from Version 2
operatorName	UDDI node ID	You must use a valid value for the UDDI node ID. This will be applied to your Version 2 data as it is migrated.
maxSearchKeys	maximum search keys	You might want to retain the value from Version 2, but can safely change this (or use the default)
getServletURLprefix	Prefix for generated discoveryURLs	You should enter a valid value for your configuration, which should therefore be the same as the value used for Version 2.
getServletName	no equivalent	Not applicable

4. Set any other properties, such as policy values, that you wish to change from the default settings (or these can be changed at a later time). For an explanation of policies and properties see "UDDI node settings" on page 515.
5. Click **Apply** to save your changes.

   Important: You cannot change mandatory node properties after initialization. If you do not save your changes before proceeding to the initialize step, you will have to delete and recreate the database.
6. After saving your changes, initialize the UDDI node by clicking **Initialize**, at the top of the pane.

   If you are migrating from Version 2 of the UDDI registry, the Version 2 data is migrated now. The initialization may take some time to complete; to track its progress, return to the node collection page and click the refresh icon at the top of the **Status** column. Alternatively, open a second administrative console window, and use the refresh icon in the same manner. The UDDI node passes through the following states

a. Initialization pending.
b. Initialization in progress.
c. Migration in progress. (This state will only occur if you are migrating.)
d. Value set creation in progress.
e. Activated.

If you have migrated the node from a previous version, return to Migrating to Version 3 of the UDDI registry to verify that the migration was successful. If you have created a new node, follow the instructions in "Using the UDDI registry Installation Verification Program (IVP)" to verify that you have successfully set up the UDDI node.

Using the UDDI registry Installation Verification Program (IVP)

This topic describes a simple test that you can carry out as an Installation Verification Program (IVP) to verify that you have deployed a UDDI registry successfully. You should perform this task *after* you have followed the instructions in Setting up and Deploying a new UDDI registry.

1. Open a browser window and enter the URL that accesses the UDDI registry User Interface (see "Displaying the UDDI registry user interface" on page 397).
2. Under the **Quick Find** heading on the **Find** tab, click the **Business** radio button and enter % in the **Starting with** field.
3. Click **Find**. If you have deployed your UDDI registry successfully, the detail frame displays the business entity which represents this UDDI node. You can click on the business entity to see its detail.

As a further installation verification test, you can publish and find more UDDI entities by using the UDDI registry User Interface, or you can compile and run one or more of the UDDI registry samples available through the UDDI registry link on the Samples for WebSphere Application Server page of the IBM developerWorks WebSphere Web site.

Changing the UDDI registry application environment after deployment

After you have deployed the UDDI registry application, you might want to change its environment. For example, you might perform initial evaluation of the UDDI registry using a Cloudscape database, and then want to put the UDDI registry into production using a DB2 database, or you might want to move from a standalone application server to a network deployment cell.

1. **Optional:** To move from a default UDDI node to a customized UDDI node, delete the UDDI registry database and recreate it by completing one of the following tasks, ensuring that you do NOT use the default node options where specified:
   - "Creating a DB2 distributed database for the UDDI registry" on page 415
   - "Creating a DB2 for z/OS database for the UDDI registry" on page 418
   - "Creating a Cloudscape database for the UDDI registry" on page 421
   - "Creating an Oracle database for the UDDI registry" on page 422

   Note: Any data saved in the default node (policies, properties and user data) will be lost when you delete the database. If you do not want to delete the database, you can instead create an entirely new customized UDDI node in a separate application server. The default UDDI node will still exist for you to use for test purposes.
2. **Optional:** To change the database type for the UDDI registry, perform the following steps:
   a. Stop the UDDI registry application (click **Applications** → **Enterprise Applications**, select the relevant check box and click **Stop**).

b. Either change the JNDI name of the existing datasource from datasources/uddids to another value, or delete the datasource. To display the datasource properties click **Resources** → **JDBC** → **JDBC providers >** *database_type* **JDBC Provider > [Additional Properties] Data sources >** *uddi_datasource*.

c. Create the new database by referring to one of the following topics:
   - "Creating a DB2 distributed database for the UDDI registry" on page 415
   - "Creating a DB2 for z/OS database for the UDDI registry" on page 418
   - "Creating a Cloudscape database for the UDDI registry" on page 421
   - "Creating an Oracle database for the UDDI registry" on page 422

d. To transfer your UDDI data, use the standard capabilities of the database products to export the data from the old database, and import it into the new one.

e. Create the new datasource. See "Creating a data source for the UDDI registry" on page 423.

f. Restart the UDDI registry application.

g. Check that you can access your UDDI data, then delete the old database.

Publishing WSDL files

To publish a Web Services Description Language (WSDL) file you need an enterprise application, also known as an enterprise archive (EAR) file, that contains a Web services-enabled module and has been deployed into WebSphere Application Server. See Deploying Web services based on Web Services for Java 2 platform, Enterprise Edition (J2EE).

The purpose of publishing the WSDL file is to provide clients with a description of the Web service, including the URL identifying the location of the service.

After installing a Web services application, and optionally modifying the endpoint information, you might need WSDL files containing the updated endpoint information. You can obtain the updated WSDL files by publishing them to the file system. If you are a client developer or a system administrator, you can use WSDL files to enable clients to connect to a Web service.

Before you publish a WSDL file, you can configure Web services to specify endpoint information in the form of URL fragments to enable full URL specification of WSDL ports. Refer to the tasks describing configuring endpoint URL information.

The WSDL files for each Web services-enabled module are published to the file system location you specify. You can provide these WSDL files to clients that want to invoke your Web services.

You can specify endpoint information for HTTP ports, Java Message Service (JMS) ports or directly access Enterprise JavaBeans (EJB) that are acting as Web services.

To publish a WSDL file:

1. Configure the URL endpoint information. Do one of the following depending on what kind of bindings you are using:
   - Configure the URL endpoint information for HTTP bindings.
   - Configure the URL endpoint information for JMS bindings.
   - Configure the URL endpoint information to directly access enterprise beans.
2. Externalize or publish the WSDL file out of the application. You can complete this task in the following ways:
   - Publish a WSDL file with the administrative console
   - Publish a WSDL file through a URL.
   - Publish a WSDL file with the **wsadmin** command tool.

Apply security to the Web service.

WSDL

Web Services Description Language (WSDL) is an Extensible Markup Language (XML)-based description language. This language was submitted to the World-Wide Web Consortium (W3C) as the industry standard for describing Web services. The power of WSDL is derived from two main architectural principles: the ability to describe a set of business operations and the ability to separate the description into two basic units. These units are a description of the operations and the details of how the operation and the information associated with it are packaged.

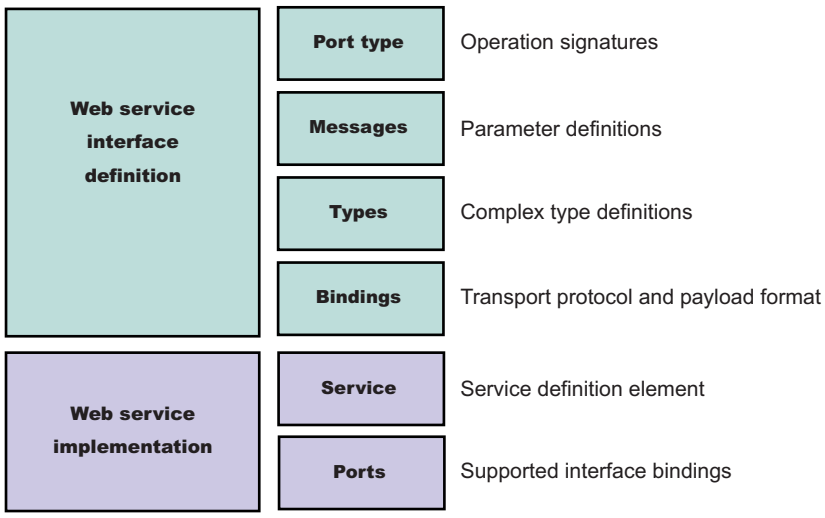
A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This separation supports the reuse of abstract definitions: messages, which are abstract descriptions of exchanged data, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Therefore, a WSDL document is composed of several elements. See WSDL architecture for more information and examples of the WSDL elements.

When creating Web services for WebSphere Application Server, you must first have an implementation bean that includes a service endpoint interface. Then, you use the **Java2WSDL** command-line tool to create a WSDL file that defines the Web services. To learn more about how the WSDL file is used in the development process, see Developing Web services.

WSDL architecture

Web Services Description Language (WSDL) files are written in Extensible Markup Language (XML). To learn more about XML, see Web services: Resources for learning.

The following is the structure of the information in a WSDL file:



A WSDL file contains the following parts:

- **Web service interface definition**
  This is part contains the elements, as well as the namespaces.
- **Web service implementation**
  This part contains the definition of the service and ports.

A WSDL file describes a Web service with the following elements:

portType

The description of the operations and associated messages. The portType element defines abstract operations.

```
<portType name="EightBall">
 <operation name="getAnswer">
  <input message="ebs:IngetAnswerRequest"/>
  <output message="ebs:OutgetAnswerResponse"/>
 </operation>
</portType>
```

message

The description of input and output parameters and return values.

```
<message name="IngetAnswerRequest">
 <part name="meth1_inType" type="ebs:questionType"/>
</message>
<message name="OutgetAnswerResponse">
 <part name="meth1_outType" type="ebs:answerType"/>
</message>
```

types

The schema for describing XML types used in the messages.

```
<types>
 <xsd:schema targetNamespace="...">
  <xsd:complexType name="questionType">
   <xsd:element name="question" type="string"/>
  </xsd:complexType>
  <xsd:complexType name="answerType">
  ...
</types>
```

binding

The bindings describe the protocol that is used to access a portType, as well as the data formats for the messages that are defined by a particular portType element.

```
<binding name="EightBallBinding" type="ebs:EightBall">
 <soap:binding style="rpc" transport="schemas.xmlsoap.org/soap/http">
 <operation name="ebs:getAnswer">
 <soap:operation soapAction="urn:EightBall"/>
  <input>
   <soap:body namespace="urn:EightBall" ... />
  ...
```

The services and ports define the location of the Web service.

Service

The service contains the Web service name and a list of ports.

Ports

The ports contain the location of the Web service and the binding used for service access.

```
<service name="EightBall">
 <port binding="ebs:EightBallBinding" name="EightBallPort">
  <soap:address location="localhost:8080/axis/EightBall"/>
 </port>
</service>
```

Multipart WSDL best practices

WebSphere Application Server supports deployment of Web services using a multipart Web Services Description Language (WSDL) file. In multipart WSDL files, an implementation WSDL file contains the wsdl:service. This implementation WSDL flie imports an interface WSDL file, which contains the other WSDL constructs. This supports multiple Web services using the same WSDL interface definition.

The <wsdl:import> element indicates a reference to another WSDL file. If the <wsdl:import> element location attribute does not contain a URL, that is, it contains only a file name, and does not begin with `http://`, `https://` or `file://`, the imported file must be located in the same directory and must not contain a relative path component. For example, if `META-INF/wsdl/A_Impl.wsdl` is in your module and contains the `<wsdl:import="A.wsdl" namespace="..."/>` import statement, the `A.wsdl` file must also be located in the module `META-INF/wsdl` directory.

It is recommended that you place all WSDL files in either the `META-INF/wsdl` directory, if you are using Enterprise JavaBeans (EJB), or the `WEB-INF/wsdl` directory, if you are using JavaBeans components, even if relative imports are located within the WSDL files. Otherwise, implications exist with the WSDL publication when you use a path like `<location="../interfaces/A_Interface.wsdl"namespace="..."/>`. Using a path like this example fails because the presence of the relative path, regardless of whether the file is located at that path or not. If the location is a Web address, it must be readable at both deployment and server startup.

WSDL publication

You can publish the files located in the `META-INF/wsdl` or the `WEB-INF/wsdl` directory through either a URL address or file, including WSDL or XML Schema Definition (XSD) files. For example, if the file referenced in the <wsdl-file> element of the `webservices.xml` deployment descriptor is located in the `META-INF/wsdl` or the `WEB-INF/wsdl` directory, it is publishable. If the files imported by the <wsdl-file> are located in the `wsdl/` directory or its subdirectory, they are publishable.

If the WSDL file referenced by the <wsdl-file> element is located in a directory other than `wsdl`, or its subdirectories, the file and its imported files, either WSDL or XSD files, which are in the same directory, are copied to the `wsdl` directory without modification when the application is installed. These types of files can also be published.

If the <wsdl-file> imports a file located in a different directory (a directory that is not `-INF/wsdl` or a subdirectory), the file is not copied to the `wsdl` directory and not available for publishing.

Configuring endpoint URL information for JMS bindings

WebSphere Application Server supports the use of the Java Message Service (JMS) API to transport Web services requests, as an alternative to using HTTP.

Review the topic Using the Java Message Service to transport Web services requests.

Configuring a service endpoint is necessary to connect Web service clients to any Web services among the components being assembled or to any external Web services. You can configure the endpoint URL information for JMS during application installation

In this task, enter the JMS endpoint URL prefix to use for each Web service-enabled Enterprise JavaBeans (EJB) Java archive (JAR) file that belong to the application. The JMS endpoint URLs are included in the Web Service Description Language (WSDL) files published for clients to use.

You can specify HTTP URL prefixes for Web services that are accessed through HTTP by using the Provide HTTP endpoint URL information panel in the administrative console. These prefixes are used to form complete endpoint addresses that are included in WSDL files when published.

You can specify JMS URL prefixes by using the Provide JMS and EJB endpoint URL information panel in the administrative console during or after application installation.

To configure JMS URL prefixes:

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Provide JMS and EJB endpoint URL information**.
3. Locate the list of Web services modules that are accessible through JMS transport.
4. Type the JMS URL fragment in the **URL fragment** field. Enter a URL fragment that is a prefix to the initial URL part that is obtained by examining the deployment information of the Web service. See the usage scenario following this task for more information.

   The value that you enter is used to define the location attribute of the port soap:address element within the WSDL file that is published using the *application_name*_ExtendedWSDLFiles.zip or the *application_name*_WSDLFiles.zip file on the Publish WSDL zip files panel.

You have a Web service that is accessible through the JMS transport and configured with JMS bindings.

Suppose an application called StockQuoteService contains an EJB JAR file that is named StockQuoteEJB, which contains one or more Web services that are accessible through the JMS transport. In Using the Java Message Service to transport Web services requests you defined a queue with the Java Naming and Directory Interface (JNDI) name of jms/StockQuote_Q, and a connection factory with the JNDI name of jms/StockQuote_CF, for your application. In this example, you specify the following string as the JMS URL prefix within the Provide JMS and EJB endpoint URL information panel:

```
jms:/queue?destination=jms/StockQuote_Q&connectionFactory=jms/StockQuote_CF
```

The WSDL publisher uses this partial URL string to produce the actual JMS URL for each port component that is defined in the module. The targetService=<*port_name*> string is added to the end of the JMS URL, for example:

```
jms:/queue?destination=jms/StockQuote_Q&connectionFactory=jms/StockQuote_CF&targetService=getQuote
```

The published WSDL file is used by clients to invoke the Web service.

Publish WSDL files.

Provide JMS and EJB endpoint URL information

Use this page to specify endpoint URL fragments for Web services accessed through SOAP and Java Message Service (JMS) or directly as enterprise JavaBean (EJBs). Fragments are used to form complete endpoint addresses included in published Web Services Description Language (WSDL) files.

To view this administrative console page, click **Applications** >**Enterprise Applications** > *application_instance* > **Provide JMS and EJB endpoint URL information**.

You can specify a fragment of the endpoint URL to be used in each Web service module. In a published WSDL file, the URL defining the target endpoint address is found in the location attribute of the port's soap:address element.

If you are using Web services modules that are configured to use JMS or configured to access EJBs directly, these modules are listed on this panel.

URL fragment for JMS

Specifies a URL fragment for Web services accessed through a JMS transport. You can enter a value that is used to define the `soap:address` of a Web service. When WSDL files are published, a URL is formed using this fragment and is contained in the WSDL files.

The URL fragment that is entered as a value is a prefix to which the targetService=property is appended to form a complete JMS URL endpoint. The default value is obtained by examining the installed service's deployment information, for example, `jms:/queue?destination=jms/MyQueue&connectionFactory=jms/MyCF`.

This information is obtained from the Web service's configured JMS endpoint, which is a Message Driven Bean (MDB) defined by the **endpointEnabler** command-line tool. You can modify the URL fragment, for example, by adding properties. The URL fragment is combined with the targetService property to form the complete URL, for example, `jms:/queue?destination=jms/MyQueue&connectionFactory=jms/MyCF &priority=5&targetService=GetQuote`.

URL fragment for EJB

Specifies a URL fragment for Web services accessed through an EJB binding. You can enter a value used to define the location attribute of the port's `generic:address` element of a Web service. This port address is contained in the WSDL zip file when the zip file is published using the ***application_name_ExtendedWSDLFiles*.zip** field on the **Publish WSDL zip file** panel.

The URL fragment value entered is a suffix, which is appended to the initial part of the URL obtained by examining the Web service's deployment information. For example, the following URL fragment can be obtained from the EJB's deployment information: `wsejb:/ com.acme.sample.MyStockQuoteHome?jndiName=ejb/MyStockQuoteHome`.

In this case, you can enter the following information in the URL fragment field, `jndiProviderURL=corbaloc:iiop:myhost.mycompany.com:2809`, which results in this endpoint URL, `wsejb:/com.acme.sample.MyStockQuoteHome?jndiName=ejb/MyStockQuoteHome &jndiProviderURL=corbaloc:iiop:myhost.mycompany.com:2809`.

Configuring the scope of a Web service port

When a Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise bean module, including implementation scope, client bindings and deployment descriptor information. There are three levels of scope that can be set: application, session and request.

Deploy the Web service into WebSphere Application Server.

Web Services for Java 2 platform Enterprise Edition (J2EE) specifies that Web services implementations must be stateless. Therefore, to maintain specification compliance, the scope can remain at the application level because the state relevant to the individual sessions level or the requests level is not supposed to be maintained in the implementation. If you want to deviate from the specification and want to access a different JavaBean instance, because you are looking for information that is located in another JavaBean implementation, the scope settings need to change.

The setting that you configure for the scope determines how frequently a new instance of a service implementation class is created for the Web service ports in a module. Use this task to configure the scope of a Web service port.

You can also configure the scope with the wsadmin tool.

To change the scope setting through the administrative console:

1. Open the administrative console.
2. Click **Applications** >**Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* >**Web Services implementation Scope**.
3. Set the scope to application, session or request. The application scope causes the same instance of the implementation to be used for all requests on the application. The session scope causes the same instance to be used for all requests in each session. The request scope causes a new instance to be used for every request. For example, with the scope set to application, every message that comes to the server accesses the same JavaBean instance because that is the way the scope settings are configured.
4. Click **Apply**.
5. Click **OK**.

The scope for a Web service port is configured.

Now you can finish any other configurations, start or stop the application, and verify the expected behavior of the Web service.

Web services implementation scope

The scope value determines when a new instance of a service implementation is created for the Web service ports in a module. When the scope value is set to application, the same instance of the implementation is used for all requests on the application. When the scope value is set to session, the same instance is used for all requests on each session. When the scope value is set to request, a new instance is created for every request.

Use this page to view and manage the scope of the ports of a Web service application.

To view this administrative console page, click **Applications** >**Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* >**Web services implementation scope**.

Related tasks

"Configuring the scope of a Web service port" on page 447
When a Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise bean module, including implementation scope, client bindings and deployment descriptor information. There are three levels of scope that can be set: application, session and request.

Port

Specifies a port name for a Web service. A module can contain one or more Web services, each of which can contain one or more ports.

Web service

Specifies the name of the Web service. A module can contain one or more Web services.

URI

Specifies the Uniform Resource Identifier (URI) of the binding file that defines the scope. The URI is relative to the Web module.

Scope

Specifies the scope of a port. The valid values for scope are request, session and application.

Configuring Web services applications with the wsadmin tool

You can use the wsadmin scripting tool to complete the several tasks for a Web services application.

Develop a Web services application.

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages.

To use the wsadmin tool to configure a Web services application or publish a Web Services Description Language (WSDL) file:

1. Launch a scripting command.
2. Follow the steps in one of the following topics, depending on what task you want to complete:
   - Configure Web service client bindings.
   - Configure Web service client preferred port mappings .
   - Configure Web service client port information .
   - Configure the scope of a Web service port .
3. Configure Web service client preferred port mappings

You have configured Web services applications with the wsadmin tool.

WSIF system management and administration

Here is an overview of where and how WSIF is installed as part of WebSphere Application Server, and a set of links to specific information about other aspects of managing your WSIF system.

The Web Services Invocation Framework (WSIF) is provided in a JAR file named `com.ibm.ws.runtime_6.1.0.jar`. The JAR file contains the core WSIF classes, and the Java, EJB, SOAP over HTTP and SOAP over JMS providers. Additional providers are packaged as separate JAR files.

When you install WebSphere Application Server, the `com.ibm.ws.runtime_6.1.0.jar` file is put on the WebSphere or Java virtual machine (JVM) class path.

WSIF requires no further configuration. WSIF is a thin abstraction layer between application code and the relevant invocation infrastructure.

For specific information about other aspects of managing your WSIF system, see the following topics:

- Maintaining the WSIF properties file
- Enabling security for WSIF
- Trace and logging for WSIF
- Troubleshooting the Web Services Invocation Framework
- WSIF messages
- WSIF - Known restrictions

Maintaining the WSIF properties file

The Web Services Invocation Framework (WSIF) properties are stored in the `com.ibm.ws.runtime_6.1.0.jar` file, in a properties file named `wsif.properties`.

The `com.ibm.ws.runtime_6.1.0.jar` file is located in the *app_server_root*`/plugins` directory, where *app_server_root* is the root directory for your installation of IBM WebSphere Application Server.

You must keep the “as shipped” `wsif.properties` file on the class path, so that WSIF can find it and the client administrator can use it to configure WSIF. However if you make any changes to the file, you do not replace the original copy in the `com.ibm.ws.runtime_6.1.0.jar` file. Instead, you save the modified version in the *app_server_root*`/lib/properties` directory.

Here is a copy of the initial contents of the `wsif.properties` file. All the possible properties are listed and described.

```
# Two properties are used to override which WSIFProvider is selected when there
# exists multiple providers supporting the same namespace URI. These properties are:
#
#    wsif.provider.default.CLASSNAME=N
#    wsif.provider.uri.M.CLASSNAME=URI
#
# CLASSNAME is the WSIFProvider class name
# N is the number of following default wsif.provider.uri.M.CLASSNAME properties
# M is a number from 1 to N to uniquely identify each wsif.provider.uri.M.CLASSNAME
#   property key.
# For example the following two properties would override the default SOAP provider
# to be the Apache SOAP provider:
#
# wsif.provider.default.org.apache.wsif.providers.soap.ApacheSOAP.WSIFDynamicProvider_ApacheSOAP=1
# wsif.provider.uri.1.org.apache.wsif.providers.soap.ApacheSOAP.WSIFDynamicProvider_ApacheSOAP=\
# http://schemas.xmlsoap.org/wsdl/soap/
#

# maximum number of milliseconds to wait for a response to a synchronous request.
# Default value if not defined is to wait forever.
# Timeout properties are only used by providers which support timeouts.
wsif.syncrequest.timeout=10000

# maximum number of seconds to wait for a response to an async request.
# if not defined on invalid defaults to no timeout
# Timeout properties are only used by providers which support timeouts.
wsif.asyncrequest.timeout=60
```

To enable your legacy Web services to continue to work with WSIF, you might need to change the default WSIF SOAP provider back to the former Apache SOAP provider.

Enabling security for WSIF

Here are the steps to be taken to enable the Web Services Invocation Framework (WSIF) to interact with a security manager.

WSIF interacts with a security manager in the following ways:

- WSIF runs in the Java 2 platform, Enterprise Edition (J2EE) security context without modification.
- When WSIF is run under a J2EE container, port implementations can use the security context to pass on security tokens or credentials as necessary.
- WSIF implementations can automatically convert J2EE security context into appropriate context for onward services.

For WSIF to interact effectively with the WebSphere Application Server security manager, complete the following step:

To load the WSDL file, enable the **FilePermission** attribute in the `was.policy` file. This permission is required when a WSDL file is referred to using the `file://` protocol.

Tips for troubleshooting the Web Services Invocation Framework

A set of specific tips to help you troubleshoot problems you experience with the Web Services Invocation Framework (WSIF).

For information about resolving WebSphere-level problems, see Diagnosing and fixing problems.

To identify and resolve Web Services Invocation Framework (WSIF)-related problems, you can use the standard WebSphere Application Server trace and logging facilities. If you encounter a problem that you think might be related to WSIF, you can check for error messages in the WebSphere Application Server administrative console, and in the application server `stdout.log` file. You can also enable the application server debug trace to provide a detailed exception dump.

A list of the WSIF run-time system messages, with details of what each message means, is provided in Message reference for WSIF.

A list of the main known restrictions that apply when using WSIF is provided in WSIF - Known restrictions.

Here is a checklist of major WSIF activities, with advice on common problems associated with each activity:

Create service

Handcrafted Web Services Description Language (WSDL) can cause numerous problems. To help ensure that your WSDL is valid, use a tool such as WebSphere Studio Application Developer (WSAD) to create your Web service. For more details about creating services with WSDL in a distributed or a z/OS environment, read the WSIF and WSDL chapter of *Developing and deploying applications* PDF book.

Define transport mechanism

For the Java Message Service (JMS), check that you have set up the Java Naming and Directory Interface (JNDI) correctly, and created the necessary connection factories and queues.

For SOAP, make sure that the deployment descriptor file `dds.xml` is correct - preferably by creating it using WebSphere Studio Application Developer (WSAD) or similar tooling.

Create client - Java code

Follow the correct format for creating a WSIF service, port, operation and message. For examples of correct code, refer to the Address Book Sample in the *Developing and deploying applications* PDF book.

Compile code (client and service)

Check that the build path against code is correct, and that it contains the correct levels of JAR files.

Create a valid EAR file for your service in preparation for deployment to a Web server.

Deploy service

When you install and deploy the service EAR file, check carefully any messages given when the service is deployed.

Server setup and start

Make sure that the WebSphere Application Server `server.policy` file (in the `/properties` directory) has the correct security settings. For more information about setting up security, see the Enabling security for WSIF topic in the *Securing applications and their environment* PDF book.

WSIF setup

Check that the `wsif.properties` file is correctly set up. For more information about this file, refer to the Maintaining the WSIF properties file topic in the *Administering applications and their environment* PDF book.

Run client

Either check that you have defined the class path correctly to include references to your client classes, WSIF JAR files and any other necessary JAR files, or (preferably) run your client using the WebSphere Application Server launchClient tool.

Either check that you have defined the class path correctly to include references to your client classes, WSIF JAR files and any other necessary JAR files, or (preferably) run your client using

the WebSphere Application Server launch client tool. For more information about this tool, refer to the Running application clients chapter of the *Developing and deploying applications* PDF book.

Here is a set of tips to help you troubleshoot commonly-experienced problems:
- "No class definition" errors received when running client code.
- "Cannot find WSDL" error.
- Your Web service EAR file does not install correctly onto the application server.
- There is a permissions problem or security error.
- Using WSIF with multiple clients causes a SOAP parsing error.
- Using the same names for JMS messaging queues and queue connection factories that run on application servers on different machines can cause JNDI lookup errors.
- A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled.
- The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not fully interoperate with services that are running on the former (Apache SOAP) provider.

"No class definition" errors received when running client code.

This problem usually indicates an error in the class path setup. Check that the relevant JAR files are included.

"Cannot find WSDL" error.

Some likely causes are:
- The application server is not running.
- The server location and port number in the WSDL are not correct.
- The WSDL is badly formed (check the error messages in the application server `stdout.log` file).
- The application server has not been restarted since the service was installed.

You might also try the following checks:
- Can you load the WSDL into your Web browser from the location specified in the error message?
- Can you load the corresponding WSDL binding files into your Web browser?

Your Web service EAR file does not install correctly onto the application server.

It is likely that the EAR file is badly formed. Verify the installation by completing the following steps:
- For an EJB binding, run the WebSphere Application Server tool `\bin\dumpnamespace`. This tool lists the current contents of the JNDI directory.
- For a SOAP over HTTP binding, open the `http://pathToServer/WebServiceName/admin/list.jsp` page (if you have the SOAP administration pages installed). This page lists all currently installed Web services.
- For a SOAP over JMS binding, complete the following checks:
  - Check that the queue manager is running.
  - Check that the necessary queues are defined.
  - Check the JNDI setup.
  - Use the "display context" option in the `jmsadmin` tool to list the current JNDI definitions.
  - Check that the Remote Procedure Call (RPC) router is running.

There is a permissions problem or security error.

Check that the WebSphere Application Server `server.policy` file (in the `/properties` directory) has the correct security settings. For more information about setting up security, see the Enabling security for WSIF topic in the *Securing applications and their environment* PDF book.

Using WSIF with multiple clients causes a SOAP parsing error.

Before you deploy a Web service to WebSphere Application Server, you must decide on the scope of the Web service. The deployment descriptor file `dds.xml` for the Web service includes the following line:

```
<isd:provider type="java" scope="Application" ......
```

You can set the `Scope` attribute to `Application` or `Session`. The default setting is `Application`, and this value is correct if each request to the Web service does not require objects to be maintained for longer than a single instance. If `Scope` is set to `Application` the objects are not available to another request during the execution of the single instance, and they are released on completion. If your Web service needs objects to be maintained for multiple requests, and to be unique within each request, you must set the scope to `Session`. If `Scope` is set to `Session`, the objects are not available to another request during the life of the session, and they are released on completion of the session. If scope is set to `Application` instead of `Session`, you might get the following SOAP error:

```
SOAPException: SOAP-ENV:ClientParsing error, response was:
FWK005 parse may not be called while parsing.;
nested exception is:

[SOAPException: faultCode=SOAP-ENV:Client; msg=Parsing error, response was:

FWK005 parse may not be called while parsing.;
        targetException=org.xml.sax.SAXException:
FWK005 parse may not be called while parsing.]
```

Using the same names for JMS messaging queues and queue connection factories that run on application servers on different machines can cause JNDI lookup errors.

You should not use the same names for messaging queues and queue connection factories that run on application servers on different machines, because WSIF always looks first for JMS destinations locally, and only uses the full JNDI reference if it cannot find the destination locally. For example, if you run a Web service on a remote machine, and have an application server running locally that uses the same names for the messaging queues and queue connection factories, then WSIF will find and use the local queues even if the remote JNDI destination is provided in full in the WSDL service definition.

A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled.

The problem appears as a log message:

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

For the steps to take to resolve the problem, see the following service integration technologies troubleshooting tip: After you migrate an application server to WebSphere Application Server Version 6, existing Web services clients can no longer use SOAP over JMS to access services hosted on the migrated server. This tip can be found in the Tips for troubleshooting service integration bus security section.

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not fully interoperable with services that are running on the former (Apache SOAP) provider.

This restriction is due to the fact that the IBM Web Service SOAP provider is designed to interoperate fully with a JAX-RPC compliant Web service, and Apache SOAP cannot provide such a service. To enable interoperation, modify either your Web service or the WSIF default SOAP provider. For information about how to modify your provider, read the chapter WSIF SOAP provider: working with legacy applications in the *Developing and deploying applications PDF* book.

Trace and logging for WSIF

The Web Services Invocation Framework (WSIF) offers trace points at the opening and closing of ports, the invocation of services, and the responses from services. WSIF also includes a `SimpleLog` utility that can run trace when you are using WSIF outside of WebSphere Application Server.

If you want to enable trace for the WSIF API within WebSphere Application Server, and have trace, stdout and stderr for the application server written to a well-known location, see Enabling tracing and logging.

To trace the WSIF API, you need to specify the following trace string:

```
wsif=all=enabled
```

To enable the WSIF `SimpleLog` utility, through which you can run trace when using WSIF outside of WebSphere Application Server, complete the following steps:

1. Create a file named `commons-logging.properties` with the following contents:

```
org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
org.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
```

2. Create a file named `simplelog.properties` with the following contents:

```
org.apache.commons.logging.simplelog.defaultlog=trace
org.apache.commons.logging.simplelog.showShortLogname=true
org.apache.commons.logging.simplelog.showdatetime=true
```

3. Put both these files, and the `commons-logging.jar` file, on the class path.

The `SimpleLog` utility writes trace to the `System.err` file.

WSIF (Web Services Invocation Framework) messages

This topic contains a list of the WSIF run-time system messages, with details of what each message means.

WebSphere system messages are logged from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message.

For more information about the message identifier format, see the topic Message reference.

WSIF0001E: An extension registry was not found for the element type "{0}"

Explanation: Parameters: {0} element type. No extension registry was found for the element type specified.

User Response: Add the appropriate extension registry to the port factory in your code.

WSIF0002E: A failure occurred in loading WSDL from "{0}"

Explanation: Parameters: {0} location of the WSDL file. The WSDL file could not be found at the location specified or did not parse correctly

User Response: Check that the location of the WSDL file is correct. Check that any network connections required are available. Check that the WSDL file contains valid WSDL.

WSIF0003W: An error occurred finding pluggable providers: {0}

Explanation: Parameters: {0} specific details about the error. There was a problem locating a WSIF pluggable provider using the J2SE 1.3 JAR file extensions to support service providers architecture. The WSIF trace file will contain the full exception details.

User Response: Verify that a META-INF/services/org.apache.wsif.spi.WSIFProvider file exists in a provider jar, that each class referenced in the META-INF file exists in the class path, and that each class implements org.apache.wsif.spi.WSIFProvider. The class in error will be ignored and WSIF will continue locating other pluggable providers.

WSIF0004E: WSDL contains an operation type "{0}" which is not supported for "{1}"

Explanation: Parameters: {0} name of the operation type specified. {1} name of the portType for the operation. An operation type which is not supported has been specified in the WSDL.

User Response: Remove any operations of the unsupported type from the WSDL. If the operation is required then make sure all messages have been correctly specified for the operation.

WSIF0005E: An error occurred when invoking the method "{1}" . ("{0}")

Explanation: Parameters: {0} name of communication type. For example EJB or Apache SOAP. {1} name of the method that failed. An error was encountered when invoking a method on the Web service using the communication shown in brackets.

User Response: Check that the method exists on the Web service and that the correct parts have been added to the operation as described in the WSDL. Network problems might be a cause if the method is remote and so check any required connections.

WSIF0006W: Multiple WSIFProvider found supporting the same namespace URI "{0}" . Found ("{1}")

Explanation: Parameters: {0} the namespace URI. {1} a list of the WSIFProvider found.. There are multiple org.apache.wsif.spi.WSIFProvider classes in the service provider path that support the same namespace URI.

User Response: A following WSIF0007I message will be issued notifying which WSIPFProvider will be used. Which WSIFProvider is chosen is based on settings in the wsif.properties file, or if not defined in the properties, the last WSIFProvider found will be used. See the wsif.properties file for more details on how to define which provider should be used to support a namespace URI.

WSIF0007I: Using WSIFProvider "{0}" for namespaceURI "{1}"

Explanation: Parameters: {0} the classname of the WSIFProvider being used. {1} the namespaceURI the provider will be used to support.. Either a previous WSIF0006W message has been issued or the SetDynamicWSIFProvider method has been used to override the provider used to support a namespaceURI.

User Response: None. See also WSIF0006W.

WSIF0008W: WSIFDefaultCorrelationService removing correlator due to timeout. ID:"{0}"

Explanation: Parameters: {0} the ID of the correlator being removed from the correlation service. A stored correlator is being removed from the correlation service due to its timeout expiring.

User Response: Determine why no response has been received for the asynchronous request within the timeout period. The wsif.asyncrequest.timeout property of the wsif.properties file defines the length of the timeout period.

WSIF0009I: Using correlation service - "{0}"

Explanation: Parameters: {0} the name of the correlation service being used. This identifies the name of the correlation service that will be used to process asynchronous requests.

User Response: None. If a correlation service other than the default WSIF supplied one is required, ensure that it is correctly registered in the JNDI java:comp/wsif/WSIFCorrelationService namespace.

WSIF0010E: Exception thrown while processing asynchronous response - "{0}"

Explanation: Parameters: {0} the error message string of the exception. While processing the response from an executeRequestResponseAsync call an exception was thrown.

User Response: Use the exception error message string to determine the cause of the error. The WSIF trace will have more details on the error including the exception stack trace.

WSIF0011I: Preferred port "{0}" was not available

Explanation: Parameters: {0} the user's preferred port. The preferred port set by the user on org.apache.wsif.WSIFService is not available

User Response: None unless this message appears for long periods of time in which case the user might want to pick a different port as their preferred port.

WSIF - Known restrictions

This topic lists the main known restrictions that apply when using WSIF.

Threading

WSIF is not thread-safe.

External Standards

WSIF supports:

- SOAP Version 1.1 (not 1.2 or later).
- WSDL Version 1.1 (not 1.2 or later).

WSIF does not provide WS-I compliance, and it does not support the Java API for XML-based Remote Procedure Calls (JAX-RPC) Version 1.1 (or later).

Full schema parsing

WSIF does not support full schema parsing. For example, WSDL references in complex types in the schema are not handled, and attributes are not handled.

XML Schema "redefine" elements are not handled and are ignored.

SOAP WSIF does not support:

- SOAP headers that are passed as `<parts>`.
- Unreferenced attachments in SOAP responses.
- Document Encoded style SOAP messages.

Note: This is not primarily a WSIF restriction. Although you can specify Document Encoded style in WSDL, it is not generally considered to be a valid option and is not supported by the Web Services Interoperability Organization (WS-I).

SOAP provider interoperability

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not fully interoperate with services that are running on the former (Apache SOAP) provider. This restriction is due to the fact that the IBM Web Service SOAP provider is designed to interoperate fully with a JAX-RPC compliant Web service, and Apache SOAP cannot provide such a service. For information on how to overcome this restriction, see WSIF SOAP provider: working with legacy applications.

WSIF's support for SOAP faults is restricted to SOAP faults originating from a Web service that runs using the IBM Web Service SOAP provider.

Note: This is not primarily a WSIF restriction. The current SOAP faults specification does not prescribe how to encode a SOAP fault so that it maps to a Java exception. Consequently, each Web service run-time environment currently decides on its own SOAP fault format. The IBM Web Service SOAP provider can understand its own response SOAP faults, but not the SOAP faults from another provider.

Type mappings

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) conforms to the JAX-RPC type mapping rules that were finalized after the former (Apache SOAP) provider was created. The majority of types are mapped the same way by both providers. The exceptions are: `xsd:date`, `xsd:dateTime`, `xsd:hexBinary` and `xsd:QName`. Both client and service need to use the same mapping rules if any of these four types are used. Below is a table detailing the mapping rules for these four types:

XML Data Type	Apache SOAP Java Mapping	JAX-RPC Java Mapping
xsd:date	java.util.Date	Not supported
xsd:dateTime	Not supported	java.util.Calendar
xsd:hexBinary	Hexadecimal string	byte [ ]
xsd:QName	org.apache.soap.util.xml.QName	javax.xml.namespace.QName

Arrays and complex types

WSIF does not support general complex types, it only handles complex types that map to Java Beans. To use schema complex types, you must write your own custom serializers. The specific complex type and array support for WSIF outbound invocation of Web services is as follows:

- WSIF supports Java classes generated by WebSphere Studio Application Developer - Integration Edition (WSAD-IE) message generators (the normal case when WSDL files are downloaded from somewhere else). The WSAD-IE-based generation happens automatically when you use the BPEL editor, or the generation actions available on the Enterprise Services context menu, or the Business Integration toolbar.
- WSIF does not support Java beans generated by other tools, including the base WSAD tool.
- For WSAD-IE generated Java beans, attributes defined in the WSDL do not work. That is to say that these attributes, although they appear in the Java beans generated to represent the complex type, do not appear in the SOAP request created by WSIF.
- WSIF does not support arrays when they are a field of a Java bean. That is to say, WSIF only supports an array that is passed in as a named `<part>`. If an array is wrappered inside a Java bean, the array is not serialized in the same way.

Object Serialization

WSIF does not support serialization of objects across different releases.

Asynchronous invocation

WSIF supports synchronous invocation for all providers. For the JMS and the SOAP over JMS providers, WSIF also supports asynchronous invocation. You should call the `supportsAsync()` method before trying to execute an asynchronous operation.

The EJB provider

The target service of the WSIF EJB provider must be a remote-home interface, it cannot be an EJB local-home interface. In addition, the EJB stub classes must be available on the client class path.

Running outside WebSphere Application Server

WSIF is not supported for use outside WebSphere Application Server.

Using the UDDI registry

Throughout this information, the term *UDDI registry* refers to the UDDI registry component that is supplied as part of WebSphere Application Server.

Select the topic you are interested in to either open documentation locally or to find information about how to locate documentation.

- Overview of the Version 3 UDDI registry
- UDDI registry terminology
- Getting started with the UDDI registry
- Migrating the UDDI registry
- Setting up and deploying a new UDDI registry
- Removing and reinstalling the UDDI registry
- Applying an upgrade to the UDDI registry
- Configuring the UDDI registry application

- Managing the UDDI registry
- UDDI registry client programming
- The UDDI registry user interface
- UDDI registry management interfaces
- Java API for XML Registries (JAXR) Provider for UDDI
- UDDI registry troubleshooting

Overview of the Version 3 UDDI registry

The Universal Description, Discovery and Integration (UDDI) specification defines a way to publish and discover information about Web services. The term 'Web service' describes specific business functionality exposed by a company, usually through an Internet connection, to allow another company, or its subsidiaries, or software program to use the service. You can find the UDDI specification on the OASIS UDDI Web page.

The UDDI specification defines a standard for the visibility, reusability and manageability essential for a Service Oriented Architecture (SOA) registry service.

The UDDI registry is a directory for Web services that is implemented using the UDDI specification. It is a component of WebSphere Application Server.

A critical component of IBM's on-demand Service Oriented Architecture, the UDDI registry solves the problem of discovery of technical components for an enterprise and its partners by:

- Providing control, flexibility and confidentiality so that an enterprise can protect its e-business investments
- Increasing efficiency by making it easier to identify technical assets
- Leveraging existing infrastructures

For example, the UDDI registry could be used in the following way within a larger enterprise:

A company has a legacy application that provides telephone numbers and Human Resources (HR) information about employees. This is turned into a Web service and published to the registry. A developer in the same company needs to write an application for a procurement function that also needs to provide HR information to the supplier. The application should allow the supplier to have access to the employee account codes once the employee provides his name or serial number. Before Web services, the developer would have been in one of the following situations:

- The developer would not have known about the similar application
- The developer would have known about the application, but been unable to reuse it due to technical barriers
- The developer would have known about the application and reused it only after significant time and negotiation

With UDDI, the developer can search for the Web service and reuse the existing technical component in their new application for the supplier in a matter of minutes. The developer saves time and gets the application up and running sooner than they would have otherwise, thereby increasing efficiency and saving the company time and money. The UDDI registry was the first version 2 standard-compliant UDDI registry for private enterprise work. The UDDI registry in this version of WebSphere Application Server builds upon previous versions and:

- Supports the UDDI Version 3.0 specification in addition to the Version 1.0 and Version 2.0 standard APIs.
- Leverages the proven, reliable WebSphere Application Server technology
- Uses a relational database, such as DB2, for its persistent store

What's new in UDDI Version 3

The main aspects of the UDDI Version 3 specification that are provided within this version of WebSphere Application Server are as follows (there are also some additional capabilities provided by the UDDI registry in WebSphere Application Server which are described in a section below) :

Improved recognition of the importance of Private UDDI Registries

These are registries that are installed, owned, managed and controlled by a separate body such as a department within a company, a company, an industry consortium or an e-marketplace.

Publisher-assigned keys

This allows the publisher of a UDDI entity to specify its key, rather than having a unique key assigned by the registry. As well as allowing more human-friendly, URI-based keys, this also makes it easier to manage multiple registries.

UDDI Information Model improvements

The UDDI data structures have been extended in a number of ways which improve the ability of UDDI to represent businesses and services via metadata.

Security Enhancements

The introduction of digital signatures provides additional security. Each of the main UDDI entities can be digitally signed, thus improving the integrity and trustworthiness of UDDI data.

Ownership transfer APIs

These allow the ownership of a UDDI entity to be transferred from one publisher to another.

UDDI Policy

Allows the behavior of a UDDI registry to be defined by setting policy, thus recognizing the various different environments in which a UDDI registry will be used.

HTTP GET support for UDDI entities

The HTTP GET service is extended beyond the scope for discovery URLs that is a part of the UDDI Version 2 specification. The service allows HTTP GET to be used to access XML representations of each of the UDDI data structures.

Additional Capabilities provided by the UDDI registry

The Version 3 UDDI registry provided in this version of WebSphere Application Server provides the following capabilities in addition to support for the UDDI Version 3 specification:

Version 2 UDDI Inquiry and Publish SOAP API compatibility

Backward compatibility is maintained for the Version 1 and Version 2 SOAP Inquiry and Publish APIs.

UDDI Administrative Console extension

The WebSphere Application Server Administrative Console includes a section which allows administrators to manage UDDI-specific aspects of their WebSphere environment. This includes the ability to set defaults for initialization of the UDDI node (such as its node ID), and to set the UDDI Version 3 Policy values.

UDDI registry Administrative Interface

A JMX administrative interface allows administrators to manage UDDI-specific aspects of the WebSphere environment programmatically.

Multi-database support

The UDDI data is persisted to a registry database. The following database products that are supported by WebSphere Application Server are also supported for use as the persistence store for the UDDI registry. Please refer to the Detailed system requirements page for specific details on supported levels.

- Cloudscape version 10.1
- DB2 version 8
- Oracle versions 9i and 10g

Documentation is provided elsewhere in this information center for setting up remote access to those databases for which this is supported.

User-defined Value Set support

This allows users to create their own categorization schemes or value sets, in addition to the standard schemes, such as NAICS, that are provided with the UDDI registry.

UDDI Utility Tools

UDDI Utility Tools allow importing and exporting of entities using the UDDI Version 2 API.

UDDI user interface

The UDDI user console supports the Inquiry and Publish APIs providing a similar level of support for the Version 3 APIs as was offered for UDDI Version 2 in WebSphere Application Server Version 5.

UDDI Version 3 Client

The Java client for UDDI Version 3 is a Java client for UDDI which handles the construction of raw SOAP requests for the client application. It is a JAX-RPC client and uses Version 3 datatypes generated from the UDDI Version 3 WSDL and schema. These datatypes are serialized or deserialized to the XML which constitutes the raw UDDI requests.

UDDI Version 2 Clients

The following clients for UDDI Version 2 requests are provided:

- UDDI4J - a Java class library for issuing UDDI requests. This was provided in WebSphere Application Server Version 5 for both UDDI Version 1 requests (uddi4j.jar) and Version 2 requests (uddi4jv2.jar). These class libraries continue to be supported, as part of the com.ibm.uddi_1.0.0.jar file, but are now both deprecated.
- JAXR - the Java API for XML Registries is a Java client API for accessing UDDI and ebXML registries. WebSphere Application Server provides a JAXR Provider for accessing the UDDI registry. It conforms to the JAXR 1.0 specification.
- EJB - an EJB interface for issuing UDDI version 2 requests. This continues to be supported but is now deprecated.

UDDI registry terminology

Throughout the UDDI documentation in this Information Center the directory location of WebSphere Application Server is referred to as *app_server_root*.

UDDI Definitions

bindingTemplate
Technical information about a service entry point and construction specifications.

businessEntity
Information about the party who publishes information about a family of services.

businessService
Descriptive information about a particular service.

Customized UDDI node
This is a UDDI node that is initialized with customized settings for the UDDI properties and UDDI policies; in particular this kind of node will have non-default values for those properties that are read-only after initialization.

A customized UDDI node is recommended for anything other than simple testing purposes (for which a default UDDI node is sufficient). You can set up a customized UDDI node by following the instructions in Setting up a customized UDDI node.

When a customized UDDI node is first started, you must set values for certain properties and then initialize the node (using the Administrative Console or UDDI Administrative Interface), before the node is ready to accept UDDI requests. The properties that need to be set control characteristics of the UDDI node that cannot be changed after initialization.

An advantage of using a customized UDDI node is that it allows you to set these properties to values that are suitable for your environment and usage of UDDI.

After a customized UDDI node has been initialized, it differs from a default UDDI node only in that it uses customized UDDI property and policy values.

Default UDDI node
This is a UDDI node which has been initialized with default settings for the UDDI properties and UDDI policies, including the properties that are read-only after initialization. A default UDDI node is intended for test purposes and as a simple way to become familiar with the behavior of the UDDI registry.

You can set up a default UDDI node in two ways. The first is to run the uddiDeploy.jacl script, specifying the 'default' option, in which case the UDDI database will be a Cloudscape database that is created for you automatically.

The second is to create the database yourself, specifying the default option, which for Cloudscape is the DEFAULT parameter when using the UDDIDerbyCreate.jar file, and for DB2 or Oracle the SQL script insert_default_database_indicator.

After a default UDDI node has been initialized, it differs from a customized UDDI node only in that it uses default UDDI property and policy values.

Policy profile
A set of UDDI policies. The default policy profile is the profile created when the default UDDI node is created. In this instance, the nodeID and root key generator are set to read only and are unchangeable after installation.

publisherAssertion
Information about a relationship between two parties, asserted by one or both.

tModel
Short for technical model.

A tModel is a data structure representing a reusable concept, such as a Web service type, a protocol used by Web services, or a category system.

tModel keys within a service description are a technical ″fingerprint″ that you can use to trace the compatibility origins of a given service. They provide a common point of reference that allows you to identify compatible services.

tModels are used to establish the existence of a variety of concepts and to point to their technical definitions. tModels that represent value sets such as category, identifier, and relationship systems are used to provide additional data to the UDDI core entities to facilitate discovery along a number of dimensions. This additional data is captured in keyedReferences that reside in category Bags, identifierBags, or publisherAssertions. The tModelKey attributes in these keyedReferences refer to the value set that relates to the concept or namespace being represented. The keyValues contain the actual values from that value set. In some cases keyNames are significant, such as for describing relationships and when using the general keywords value set. In all other cases, however, keyNames are used to provide a human readable version of what is in the keyValue.

UDDI Application
The UDDI registry J2EE application.

UDDI entitlement
An entitlement that a UDDI user or publisher has within a UDDI registry, such as the capability to publish keyGenerators, or the tier to which the publisher is assigned (in other words, the number of entities that the publisher is entitled to publish). Each UDDI publisher will have a range of settings for the various UDDI entitlements. A UDDI entitlement is sometimes referred to as a 'user entitlement', or as the UDDI publisher's set of 'user entitlements'.

UDDI Node
A set of Web Services supporting at least one of the UDDI API sets, which supports interaction with UDDI data through the UDDI APIs. There is no direct mapping between a UDDI node and a WebSphere Application Server node. A UDDI node consists of an instance of the UDDI application running in an application server (or a cluster of UDDI application instances running in a cluster of application servers) together with an instance of the UDDI database containing UDDI data.

UDDI node initialization
The process of node initialization sets up values in the UDDI database, and establishes the ″personality″ of the UDDI node. A UDDI node cannot accept UDDI API requests until it has been initialized.

UDDI node state
Describes the current state of the UDDI node, as opposed to the state of the UDDI application (which is either stopped or started). The state of a UDDI node can be one of not initialized, initialization pending, initialization in progress, migration pending, migration in progress, value set creation pending, value set creation in progress, activated or deactivated.

UDDI NodeId
A unique identifier of a UDDI node.

UDDI Policy
A UDDI policy is a statement of required and expected behavior of a UDDI registry, specified via policy values for the various policies defined in the UDDI Version Specification.

UDDI property
A value for a property which controls the personality or behavior of a UDDI node.

UDDI publisher
A WebSphere user who is entitled to publish UDDI entities to a specified UDDI registry. A UDDI publisher is sometimes referred to as a 'UDDI user', or simply as a 'publisher' when used in a UDDI context.

UDDI registry
A UDDI registry comprises one or more UDDI nodes. The UDDI registry in this version of WebSphere Application Server supports single-node UDDI registries only.

UDDI Tier
Determines the number of UDDI entities of each type (business, services per business, bindings per service, tModel, publisher assertion) that a UDDI publisher is entitled to publish. Each UDDI publisher will be assigned (either by default or explicitly by a UDDI administrator) to a particular tier, and will not be able to publish more entities than are allowed for that tier. There are some predefined tiers supplied with the UDDI registry, and a UDDI administrator can create additional tiers. A UDDI tier is often referred to simply as a 'tier' when used in a UDDI context.

Version 2 UDDI registry
A shorthand term used to refer to an UDDI registry implementation which supports Version 2 of the UDDI specification (and also Version 1). A Version 2 UDDI registry is included in WebSphere Application Server Network Deployment Version 5.x.

Version 3 UDDI registry
A shorthand term used to refer to an UDDI registry implementation which supports Version 3 of the UDDI specification (and also Versions 1 and 2). A Version 3 UDDI registry is included in WebSphere Application Server. It should be noted that the term 'Version 3 UDDI registry' does not indicate a registry which only supports UDDI version 3 requests.

The following table shows how the various versions of the UDDI registry relate to the relevant OASIS specification and WebSphere Application Server level:

UDDI registry Version	OASIS UDDI specification levels supported	Supported on WebSphere Application Server version
1.1	• UDDI Version 1 • UDDI Version 2	4.0.2
1.1.1	• UDDI Version 1 • UDDI Version 2	4.0.3 and later
2.0.x	• UDDI Version 1 • UDDI Version 2	5.0.x
2.1.x	• UDDI Version 1 • UDDI Version 2	5.1.x
3.0.2	• UDDI Version 1 • UDDI Version 2.0.4 (APIs), Version 2.0.3 (data structures) • UDDI Version 3.0.2	6.0 and later

UDDI registry management interfaces

This topic explains interfaces and tools that you can use to manage UDDI nodes programmatically.

UDDI registry Administrative (JMX) Interface

The UDDI registry Administrative (JMX) Interface provides a Java API that allows you to manage runtime configuration settings to control UDDI registry runtime behavior, such as setting the maximum number of results that UDDI users can receive for inquiry requests, or creating publish limits for UDDI publishers. Sample client code is provided for you to build on.

User Defined Value Set Support in the UDDI registry

User Defined Value Set Support in the UDDI registry explains the tooling provided to manage your own categorization value sets, including loading value set data into a UDDI registry node.

UDDI Utility Tools

UDDI Utility Tools explains the tooling and Java API for promoting version 2 entities from one UDDI registry to another while retaining entity keys. This is particularly useful for publishing canonical tModels with a predefined key.

UDDI registry Administrative (JMX) Interface

Each WebSphere UDDI registry application registers an MBean with an MBean identifier of 'UddiNode'. This MBean may be used by client applications to inspect and manage the runtime configuration of a UDDI application. This includes managing the activation state of and information about a UDDI node, updating properties and policies, setting publish tier limits, registration of UDDI publishers, and controlling value set support.

You can read and invoke the UddiNode attributes and operations using standard JMX interfaces. A client utility class UddiNodeProxy.java provides a ready-made application to connect to a UddiNode MBean and perform all the available operations. Example classes are also provided to drive UddiNodeProxy and demonstrate how to use the various UDDI management data types.

When WebSphere Application Server security is enabled, you can only invoke the operations of the UddiNode MBean if you are a user in an administrative role. The operations which make updates require the Administrator or Operator role, while get operations can be performed by Administator, Operator, Configurator and Monitor roles.

UddiNodeProxy Usage

The following .jar files are required for compilation:

- *app_server_root*/plugins/com.ibm.uddi_1.0.0.jar
- *app_server_root*/runtimes/com.ibm.ws.admin.client_6.1.0.jar

The UddiNodeProxy class provides a utility method to programmatically interrogate the UddiNode MBean and output all the available attributes, operations and notifications to System.out. For each operation, the return type, operation name and parameter types are output as well as the impact property which indicates how the operation changes the state of the UddiNode MBean (and the UDDI node). As for all MBeans, the value for the impact property can be one of:

ACTION:
state of MBean will be changed

INFO: of the MBean remains unchanged and will return information

ACTION_INFO:
state of the MBean will change and return some information

UNKNOWN:
the impact of invoking the operation is not known

1. Invoke outputMBeanInterface:`uddiNode.outputMBeanInterface();`

   Expected output:

```
java.lang.String getNodeID() [INFO]
(getter for attribute nodeID)
java.lang.String getNodeState() [INFO]
(getter for attribute nodeState)
java.lang.String getNodeDescription() [INFO]
(getter for attribute nodeDescription)
java.lang.String getNodeApplicationName() [INFO]
(getter for attribute nodeApplicationName)
void activateNode() [ACTION]
(activates UDDI node)
void deactivateNode() [ACTION]
```

```
(deactivates UDDI node)
void initNode() [ACTION]
(initializes Uddi node)
com.ibm.uddi.v3.management.Property getProperty(java.lang.String propertyId) [INFO]
(returns UDDI Property)
com.ibm.uddi.v3.management.PolicyGroup getPolicyGroup(java.lang.String policyGroupId) [INFO]
(returns UDDI PolicyGroup)
com.ibm.uddi.v3.management.Policy getPolicy(java.lang.String policyId) [INFO]
(returns UDDI Policy)
void updatePolicy(com.ibm.uddi.v3.management.Policy policy) [ACTION]
(updates UDDI Policy)
void updateProperty(com.ibm.uddi.v3.management.ConfigurationProperty property) [ACTION]
(updates UDDI Property)
void updateProperties(java.util.List properties) [ACTION]
(updates collection of UDDI properties)
void updatePolicies(java.util.List policies) [ACTION]
(updates collection of UDDI policies)
java.util.List getProperties() [INFO]
(returns the collection of UDDI properties)
java.util.List getPolicyGroups() [INFO]
(returns collection of policy groups (note that the policies are not populated))
java.util.List getValueSets() [INFO]
(returns collection of value set status objects)
com.ibm.uddi.v3.management.ValueSetStatus getValueSetDetail(java.lang.String tModelKey) [INFO]
(returns status for a value set)
com.ibm.uddi.v3.management.ValueSetProperty getValueSetProperty(java.lang.String tModelKey,java.lang.
String valueSetPropertyId) [INFO]
(returns a property of a value set)
void updateValueSet(com.ibm.uddi.v3.management.ValueSetStatus valueSet) [ACTION]
(updates value set status)
void updateValueSets(java.util.List valueSets) [ACTION]
(updates multiple value sets)
void loadValueSet(java.lang.String filePath,java.lang.String tModelKey) [ACTION]
(loads values for a value set from a UDDI registry V3/V2 taxonomy data file.)
void loadValueSet(com.ibm.uddi.v3.management.ValueSetData valueSetData) [ACTION]
(loads values for a value set with the given tModel key.)
void changeValueSetTModelKey(java.lang.String oldTModelKey,java.lang.String newTModelKey) [ACTION]
(replaces all occurrences of values belonging to original tModelKey to new tModelKey.)
void unloadValueSet(java.lang.String tModelKey) [ACTION]
(unloads values for a value set with the given tModel key.)
java.lang.Boolean isExistingValueSet(java.lang.String tModelKey) [INFO]
(Determine if Value Set data exists for the given tModel key.)
java.util.List getTierInfos() [INFO]
(returns the collection of UDDI tier descriptions.)
java.util.List getLimitInfos() [INFO]
(returns the collection of UDDI limit descriptions.)
java.util.List getEntitlementInfos() [INFO]
(returns the collection of UDDI entitlements.)
com.ibm.uddi.v3.management.Tier getTierDetail(java.lang.String tierId) [INFO]
(returns UDDI Tier detail, specifying limits to the number of entities that can be published.)
com.ibm.uddi.v3.management.Tier createTier(com.ibm.uddi.v3.management.Tier tier) [ACTION]
(creates a UDDI Tier, specifying limits to the number of entities that can be published. Returns the
new tier ID.)
com.ibm.uddi.v3.management.Tier updateTier(com.ibm.uddi.v3.management.Tier tier) [ACTION]
(updates UDDI Tier details. Returns the updated Tier.)
void deleteTier(java.lang.String tierId) [ACTION]
(deletes the UDDI Tier, if it not in use.)
void setDefaultTier(java.lang.String tierId) [ACTION]
(Specifies the tier that auto registered UDDI publishers are assigned to.)
java.lang.Integer getUserCount(java.lang.String tierId) [INFO]
(returns the number of UDDI publisher within the specified tier.)
com.ibm.uddi.v3.management.TierInfo getUserTier(java.lang.String userId) [INFO]
(returns UDDI Tier information, specifying the tier this user belongs to.)
com.ibm.uddi.v3.management.UddiUser getUddiUser(java.lang.String userId) [INFO]
(returns UDDI user details, including tier and entitlements details.)
java.util.List getUserInfos() [INFO]
(returns the collection of UDDI user names and the tier they belong to.)
```

```
void createUddiUser(com.ibm.uddi.v3.management.UddiUser user) [ACTION]
(creates a new UDDI user.)
void createUddiUsers(java.util.List users) [ACTION]
(creates the collection of new UDDI users.)
void updateUddiUser(com.ibm.uddi.v3.management.UddiUser user) [ACTION]
(updates UDDI user details.)
void deleteUddiUser(java.lang.String userId) [ACTION]
(deletes UDDI publisher.)
void assignTier(java.util.List userIds,java.lang.String tierId) [ACTION]
(sets the tier for a List of users.)
notificationInfo: description=default UDDI event,descriptorType=notification,severity=(6),name=
uddi.node.event
notificationInfo: description=null,descriptorType=notification,severity=(6),name=jmx.attribute.
changed
```

See ManageNodeInfoSample class for sample code that demonstrates the attributes and operations described in this section.

Managing UDDI Node States and Attributes

UDDI nodes can be in one of several states, depending on the way the UDDI application was installed (as a default configuration or one where the administrator controls when initialization occurs). The UddiNode MBean provides four read only attributes: nodeID, nodeState, nodeDescription and nodeApplicationName. In addition the following MBean operations change UDDI node state: activateNode, deactivateNode and initNode.

nodeID

The node ID is the unique identifier for a UDDI node. If the UDDI application is installed as a default configuration the node ID is automatically generated. If the UDDI application is set up manually, the node ID is set by the administrator. It must be a valid UDDI key.

```
String nodeID = uddiNode.getNode();

System.out.println("node ID: " + nodeId);
```

nodeState

The nodeState attribute can have one of the following values:

nodeState value	English text associated with state
node.state.uninitialized	Not initialized
node.state.initialized	Initialized
node.state.initPending	Initialization pending
node.state.initInProgress	Initialization in progress
node.state.initMigrationPending	Migration pending
node.state.initMigration	Migration in progress
node.state.initValueSetCreationPending	Value set creation pending
node.state.initValueSetCreation	Value set creation in progress
node.state.activated	Activated
node.state.deactivated	Deactivated
node.state.unknown	Unknown

After installing a UDDI application using the default configuration, the UDDI node will be in activated state, that is, ready to receive and process UDDI API requests. The node ID and root key generator and some

other properties are generated and cannot be changed. For a manually installed UDDI application where you want to specify the UDDI node ID and root key generator values, starting the UDDI application will put the UDDI node into initPending state. In this state, you can update all writable values up until the point you invoke the initNode operation. The initNode operation loads base tModels and value set data and writes all the configuration data to the UDDI node's database. During initialization the state is initInProgress. When initialization completes, the state changes momentarily to initialized and settles at activated. At this point the state can only be switched between activated and deactivated using deactivateNode and activateNode MBean operations.

Each node state value is in fact a message key which can be looked up in the messages.properties resource bundle. The attribute value can be retrieved using the getNodeState method of UddiNodeProxy:

1. Invoke getNodeState:

```
String nodeStateKey = uddiNode.getNodeState();
```

2. Look up translated text from ResourceBundle and output:

```
String messages = "com.ibm.uddi.v3.management.messages";

ResourceBundle bundle = ResourceBundle.getBundle(messages,
                                                 Locale.ENGLISH);

String nodeStateText =  bundle.getString(nodeStateKey);

System.out.println("node state: " + nodeStateText);
```

nodeDescription

You can get the administrator assigned description for the UDDI node using the getNodeDescription method of UddiNodeProxy:

1. Invoke getNodeDescription and output:

```
String nodeDescription = uddiNode.getNodeDescription();
System.out.println("node description: " + nodeDescription);
```

nodeApplicationName

The nodeApplicationName attribute is useful for discovering where the UDDI application that corresponds to the UDDI node is installed. The value will be a concatenation of the cell, node and server names, separated by colons. Retrieve the application location using the getApplicationId method of UddiNodeProxy:

1. Invoke getApplicationId and output:

```
String nodeApplicationId = uddiNode.getApplicationId();

System.out.println("node application location: " +
                                          nodeApplicationId);
```

activateNode

Changes the state of the UDDI node to activated, if the UDDI node was previously deactivated.

1. . Invoke activateNode:

```
uddiNode.activateNode();
```

deactivateNode

Changes the state of the UDDI node to deactivated, if the UDDI node was previously activated.

1. Invoke deactivateNode:

```
uddiNode.deactivateNode();
```

initNode

Causes UDDI node initialization, and when this completes the state of the UDDI node is 'activated'.

1. Invoke initNode:

```
uddiNode.initNode();
```

Managing Configuration Properties

UDDI node runtime behavior is affected by the setting of several configuration properties. The UddiNode MBean provides operations to inspect and update their values, as follows: getProperties, getProperty, updateProperty and updateProperties.

See ManagePropertiesSample class for sample code that demonstrates the operations described in this section.

getProperties

Returns collection of all configuration properties as ConfigurationProperty objects.

1. Invoke getProperties:

```
List properties = uddiNode.getProperties();
```

2. Cast each collection member to ConfigurationProperty:

```
if (properties != null) {
  for (Iterator iter = properties.iterator(); iter.hasNext();) {
       ConfigurationProperty property =
                        (ConfigurationProperty) iter.next();
       System.out.println(property);
  }
}
```

Once you have the ConfigurationProperty objects you can inspect attributes like the ID, value, type, whether the property is read only, required for initialization, and get name and description message keys. For example, invoking the toString method returns results similar to:

```
ConfigurationProperty
id: operatorNodeIDValue
nameKey: property.name.operatorNodeIDValue
descriptionKey: property.desc.operatorNodeIDValue
type: java.lang.String
value: uddi:capnscarlet:capnscarlet:server1:default
unitsKey:
readOnly: true
required: true
usingMessageKeys: false
validValues: none
```

You can use the nameKey and descriptionKey values to look up the translated name and description for a given locale, using the messages.properties resource in the sample package.

getProperty

Returns ConfigurationProperty object with the specified ID. Available property IDs are specified in PropertyConstants together with descriptions of the purpose of the corresponding properties.

1. Invoke getProperty:

```
ConfigurationProperty property =
uddiNode.getProperty(PropertyConstants.DATABASE_MAX_RESULT_COUNT);
```

2. To retrieve the value of the property you could use the getValue method which returns an Object, but in this case, the property is of type integer, so it's easier to retrieve the value using the convenience method getIntegerValue:

```
int maxResults = property.getIntegerValue();
```

updateProperty

Updates the value of the ConfigurationProperty object with the specified ID. Available property IDs are specified in PropertyConstants together with descriptions of the purpose of the corresponding properties. Although you can invoke the setter methods in a ConfigurationProperty object, the only value that is updated in the UDDI node is the value. So to update a property, the steps are typically:

1. Create a ConfigurationProperty object and set its ID:

```
ConfigurationProperty defaultLanguage = new ConfigurationProperty();
defaultLanguage.setId(PropertyConstants.DEFAULT_LANGUAGE);
```

2. Set the value:

```
defaultLanguage.setStringValue("ja");
```

3. Invoke updateProperty:

```
uddiNode.updateProperty(defaultLanguage);
```

updateProperties

Updates several ConfigurationProperty objects in a single request. Set up the ConfigurationProperty objects as for the updateProperty operation.

1. Add updated properties to a List:

```
List updatedProperties = new ArrayList();

updatedProperties.add(updatedProperty1);
updatedProperties.add(updatedProperty2);
```

2. Invoke updateProperties:

```
uddiNode.updateProperties(updatedProperties);
```

Managing Policies

Policies affecting behavior of the UDDI API are managed using the following UddiNode operations: getPolicyGroups, getPolicyGroup, getPolicy, updatePolicy and updatePolicies.

See ManagePoliciesSample class for sample code that demonstrates the attributes and operations described in this section.

getPolicyGroups

Returns collection of all policy groups as PolicyGroup objects.

1. Invoke getPolicyGroups:

```
List policyGroups = uddiNode.getPolicyGroups();
```

2. Cast each collection member to PolicyGroup:

```
if (policyGroups != null) {
for (Iterator iter = policyGroups.iterator(); iter.hasNext();) {
  PolicyGroup policyGroup = (PolicyGroup) iter.next();
  System.out.println(policyGroup);
  }
}
```

Each policy group has an ID, name and description key, which you can look up in the messages.properties resource in the sample package. Although the PolicyGroup class does have a getPolicies method, PolicyGroup objects that are returned by the getPolicyGroups operation do not contain any Policy objects. Because of this behavior, clients can determine the known policy groups, and their IDs, without retrieving the entire set of policies in one request. To retrieve the policies within a policy group, use the getPolicyGroup operation.

getPolicyGroup

Returns the PolicyGroup object with the supplied ID.

1. Convert policy group ID to a String:

```
String groupId = Integer.toString(PolicyConstants.REG_APIS_GROUP);
```

2. Invoke getPolicyGroup:

```
PolicyGroup policyGroup = uddiNode.getPolicyGroup(groupId);
```

getPolicy

Returns the Policy object for the specified ID. Like a ConfigurationProperty, a Policy object has an ID, name and description keys, type, value and indicators specifying if the policy is read only or required for node initialization.

1. Convert policy ID to a String:

```
String policyId = Integer.toString(
            PolicyConstants.REG_AUTHORIZATION_FOR_INQUIRY_API);
```

2. Invoke getPolicy:

```
Policy policy = uddiNode.getPolicy(policyId);
```

updatePolicy

Updates the value of the Policy object with the specified ID. Available policy IDs are specified in PolicyConstants together with descriptions of the purpose of the corresponding policies. Although you can invoke the setter methods in a Policy object, the only value that is updated in the UDDI node is the value. So to update a policy, the steps are typically:

1. Create a Policy object and set its ID:

```
Policy updatedPolicy = new Policy();
String policyId =
      Integer.toString(PolicyConstants.REG_SUPPORTS_UUID_KEYS);
updatedPolicy.setId(policyId);
```

2. Set the value:

```
updatedPolicy.setBooleanValue(true);
```

3. Invoke updatePolicy:

```
uddiNode.updatePolicy(updatedPolicy);
```

updatePolicies

Updates several Policy objects in a single request. Set up the Policy objects as for the updatePolicy operation.

1. Add updated policies to a List:

```
List updatedPolicies = new ArrayList();

updatedPolicies.add(updatedPolicy1);
updatedPolicies.add(updatedPolicy2);
```

2. Invoke updatePolicies:

```
uddiNode.updatePolicies(updatedPolicies);
```

Managing Tiers

Tiers control how many of each type of UDDI entities a publisher can save in the UDDI registry. A tier has an ID, an administrator defined name and description, and a set of limits, one for each type of entity. Tiers are managed using the following UddiNode operations: createTier, getTierDetail, getTierInfos, getLimitInfos, setDefaultTier, updateTier, deleteTier and getUserCount.

See ManageTiersSample class for sample code that demonstrates the attributes and operations described in this section.

createTier

Creates a new tier, with specified publish limits for each UDDI entity.

1. Set tier name and description in a TierInfo object.

```
String tierName = "Tier 100";
String tierDescription = "A tier with all limits set to 100.";

TierInfo tierInfo = new TierInfo(null, tierName, tierDescription);
```

2. Define Limit objects for each UDDI entity:

```
 List limits = new ArrayList();

Limit businessLimit = new Limit();
businessLimit.setIntegerValue(100);

businessLimit.setId(LimitConstants.BUSINESS_LIMIT);

Limit serviceLimit = new Limit();
serviceLimit.setIntegerValue(100);
serviceLimit.setId(LimitConstants.SERVICE_LIMIT);

Limit bindingLimit = new Limit();
bindingLimit.setIntegerValue(100);
bindingLimit.setId(LimitConstants.BINDING_LIMIT);

Limit tModelLimit = new Limit();
tModelLimit.setIntegerValue(100);
tModelLimit.setId(LimitConstants.TMODEL_LIMIT);

Limit assertionLimit = new Limit();
assertionLimit.setIntegerValue(100);

assertionLimit.setId(LimitConstants.ASSERTION_LIMIT);
limits.add(businessLimit);
limits.add(serviceLimit);
limits.add(bindingLimit);
limits.add(tModelLimit);
limits.add(assertionLimit);
```

3. Create Tier object:

```
  Tier tier = new Tier(tierInfo, limits);
```

4. Invoke create Tier and retrieve created tier:

```
Tier createdTier = uddiNode.createTier(tier);
```

5. Inspect generated tier ID of created tier:

```
tierId = createdTier.getId();
System.out.println("created tier has ID: " + tierId);
```

getTierDetail

Returns the Tier object for the given tier ID. The Tier class has getter methods for the tier ID, tier name and description (as set by the administrator), and the collection of Limit objects which specify how many of each UDDI entity type may be published by UDDI publishers allocated to the tier. The isDefault method indicates whether the tier is the default tier, that is, the tier that is allocated to UDDI publishers when auto registration is enabled.

1. Invoke getTierDetail:

```
 Tier tier = uddiNode.getTierDetail("2");
```

updateTier

Updates tier contents with the supplied Tier object.

1. Update an existing Tier object (which may have been newly instantiated, or returned by the getTierDetail or createTier operations). This example retains the tier name and description, and all the limit values except the limit being updated:

```
modifiedTier.setName(tier.getName());
modifiedTier.setDescription(tier.getDescription());

Limit tModelLimit = new Limit();
tModelLimit.setId(LimitConstants.TMODEL_LIMIT);
tModelLimit.setIntegerValue(50);

List updatedLimits = new ArrayList();
updatedLimits.add(tModelLimit);

modifiedTier.setLimits(updatedLimits);
```

2. Invoke updateTier:

```
uddiNode.updateTier(modifiedTier);
```

getTierInfos

Returns collection of lightweight tier descriptor objects (TierInfo) which contain the tier ID, and tier name and description values, and whether the tier is the default tier.

1. Invoke getTierInfos:

```
List tierInfos = uddiNode.getTierInfos();
```

2. Output content of each TierInfo:

```
if (tierInfos != null) {

  for (Iterator iter = tierInfos.iterator(); iter.hasNext();) {
     TierInfo tierInfo = (TierInfo) iter.next();
     System.out.println(tierInfo);
  }
}
```

setDefaultTier

Specifies the tier with the given tier ID is the default tier. The default tier is the tier that is allocated to UDDI publishers when auto registration is enabled. Typically this would be set to a tier with low publish limits to prevent casual users publishing too many entities.

1. Invoke setDefaultTier:

```
uddiNode.setDefaultTier("4");
```

deleteTier

Removes the tier with the given tier ID. Tiers can only be removed if they have no UDDI publishers assigned to them, and the tier is not the default tier.

1. Invoke deleteTier:

```
uddiNode.deleteTier("4");
```

getUserCount

Returns the number of UDDI publishers assigned to tier specified by the tier ID.

1. Invoke getUserCount:

```
Integer userCount = uddiNode.getUserCount("4");
System.out.println("users in tier 4: " + userCount.intValue());
```

getLimitInfos

Returns collection of Limit objects representing the limit values for each type of UDDI entity. Limits are used in Tier objects.

1. Invoke getLimitInfos:

```
List limits = uddiNode.getLimitInfos();
```

2. Output the ID and limit value for each Limit object:

```
for (Iterator iter = limits.iterator(); iter.hasNext();) {
  Limit limit = (Limit) iter.next();

  System.out.println("limit ID: "
                     + limit.getId()
                     + ", limit value: "
                     + limit.getIntegerValue());
}
```

Managing UDDI Publishers

UDDI publishers are managed using the UddiNode MBean operations createUddiUser, createUddiUsers, updateUddiUser, deleteUddiUser, getUddiUser, getUserInfos, getEntitlementInfos, assignTier, getUserTier. An example is provided for each, making use of the UddiNodeProxy client class.

See ManagePublishersSample class for sample code that demonstrates the attributes and operations described in this section.

createUddiUser

Registers a single UDDI publisher, in a specified tier, with specified entitlements. The UddiUser class represents the UDDI publisher, and this is constructed using a user ID, a TierInfo object which specifies the tier ID to allocate the UDDI publisher to, and a collection of Entitlement objects which specify what the UDDI publisher is permitted to do.

Tip: to allocate the UDDI publisher default entitlements, set the entitlements parameter to null.

1. Create the UddiUser object:

```
UddiUser user = new UddiUser("user1", new TierInfo("3"), null);
```

2. Invoke createUddiUser:

```
uddiNode.createUddiUser(user);
```

createUddiUsers

Registers multiple UDDI publishers. This example shows how to register 7 UDDI publishers in one call, with default entitlements.

1. Create TierInfo objects for tiers that publishers will be allocated to:

```
TierInfo tier1 = new TierInfo("1");
TierInfo tier4 = new TierInfo("4");
```

2. Create UddiUser objects for each UDDI publisher, specifying tier to allocate to:

```
UddiUser publisher1 = new UddiUser("Publisher1", tier4, null);
UddiUser publisher2 = new UddiUser("Publisher2", tier4, null);
UddiUser publisher3 = new UddiUser("Publisher3", tier4, null);
UddiUser publisher4 = new UddiUser("Publisher4", tier1, null);
UddiUser publisher5 = new UddiUser("Publisher5", tier1, null);
UddiUser cts1 = new UddiUser("cts1", tier4, null);
UddiUser cts2 = new UddiUser("cts2", tier4, null);
```

3. Add the UddiUser objects to a List:

```
List uddiUsers = new ArrayList();

uddiUsers.add(publisher1);
uddiUsers.add(publisher2);
```

```
uddiUsers.add(publisher3);
uddiUsers.add(publisher4);
uddiUsers.add(publisher5);
uddiUsers.add(cts1);
uddiUsers.add(cts2);
```

4. Invoke createUddiUsers:

```
uddiNode.createUddiUsers(uddiUsers);
```

updateUddiUser

Updates a UDDI publisher with the details in the supplied UddiUser object. This is typically used to change the tier of one UDDI publisher or update their entitlements. Tip: only supply the entitlements you want to update – the remainder of available entitlements will retain their existing value.

1. Create Entitlement objects with appropriate permission. (the entitlement IDs are found in EntitlementConstants:

```
Entitlement publishUuiDKeyGenerator =
    new Entitlement(PUBLISH_UUID_KEY_GENERATOR, true);
Entitlement publishWithUuidKey =
    new Entitlement(PUBLISH_WITH_UUID_KEY, true);
```

2. Add Entitlement objects to a List:

```
List entitlements = new ArrayList();
entitlements.add(publishUuiDKeyGenerator);
entitlements.add(publishWithUuidKey);
```

3. Update a UddiUser object with the updated entitlements:

```
user.setEntitlements(entitlements);
```

4. Invoke updateUddiUser:

```
uddiNode.updateUddiUser(user);
```

getUddiUser

Retrieves details about a UDDI publisher in the form of a UddiUser object. This specifies the UDDI publisher ID, information about the tier they are assigned to and the entitlements they possess.

1. Invoke getUddiUser:

```
UddiUser user1 = uddiNode.getUddiUser("user1");
```

2. Output the contents of UddiUser:

```
System.out.println("retrieved user: " + user1);
```

getUserInfos

Returns a collection of UserInfo objects. Each UserInfo represents a UDDI publisher known to the UDDI node, and the name of the tier they are allocated to. To get details about a specific UDDI publisher, including the tier ID, and entitlements, use the getUddiUser operation.

1. Invoke getUserInfos:

```
List registeredUsers = uddiNode.getUserInfos();
```

2. Output the UserInfo objects:

```
System.out.println("retrieved registered users: ");
System.out.println(registeredUsers);
```

getEntitlementInfos

Returns a collection of Entitlement objects. Each entitlement is a property that controls whether permission is granted to a UDDI publisher to perform a specified action.

1. Invoke getEntitlementInfos:

```
List entitlementInfos = uddiNode.getEntitlementInfos();
```

2. Specify where to find message resources:

```
String messages = "com.ibm.uddi.v3.management.messages";
ResourceBundle bundle = ResourceBundle.getBundle(
                                    messages, Locale.ENGLISH);
```

3. Iterate through the Entitlement objects, displaying the ID, name and description:

```
for (Iterator iter = entitlementInfos.iterator(); iter.hasNext();) {
  Entitlement entitlement = (Entitlement) iter.next();

  StringBuffer entitlementOutput = new StringBuffer();

  String entitlementId = entitlement.getId();
  String entitlementName =
              bundle.getString(entitlement.getNameKey());
String entitlementDescription =
              bundle.getString(entitlement.getDescriptionKey());

  entitlementOutput.append("Entitlement id: ");
  entitlementOutput.append(entitlementId);
  entitlementOutput.append("\n  name: ");
  entitlementOutput.append(entitlementName);
  entitlementOutput.append("\n  description: ");
  entitlementOutput.append(entitlementDescription);

  System.out.println(entitlementOutput.toString());
}
```

deleteUddiUser

Removes the UDDI publisher with the specified user name (ID) from the UDDI registry.

1. Invoke deleteUddiUser:

```
uddiNode.deleteUddiUser("user1");
```

assignTier

Assigns UDDI publishers with supplied IDs to the specified tier. This is useful when you want to restrict several UDDI publishers, perhaps by assigning them all to a tier that doesn't allow publishing of any entities.

1. Create list of publisher IDs:

```
List uddiUserIds = new ArrayList();

uddiUserIds.add("Publisher1");
uddiUserIds.add("Publisher2");
uddiUserIds.add("Publisher3");
uddiUserIds.add("Publisher4");
uddiUserIds.add("Publisher5");
uddiUserIds.add("cts1");
uddiUserIds.add("cts2");
```

2. Invoke assignTier:

```
uddiNode.assignTier(uddiUserIds, "0");
```

getUserTier

Returns information about the tier a UDDI publisher is assigned to. The returned TierInfo has getters methods for retrieving the tier ID, tier name, tier description, and whether the tier is the default tier.

1. Invoke getUserTier:

```
TierInfo tierInfo = getUserTier("Publisher3");
```

2. Output the contents of the TierInfo object:

```
System.out.println(tierInfo);
```

Managing Value Sets

Value sets are represented in a UDDI registry as value set tModels, with a UDDI types keyedReference with value 'categorization'. Such value sets are backed with a set of valid values and for user defined value sets, this data is loaded into the UDDI registry using UddiNode MBean operations (although it is more convenient to use the User defined value set tool for this purpose). Each value set can be controlled by policy as being supported or not supported. When a value set is supported by policy, it can be referenced within UDDI publish requests. The UddiNode operations available to manage value sets and their data are: getValueSets, getValueSetDetail, getValueSetProperty, updateValueSet, updateValueSets, loadValueSet, changeValueSetTModelKey, unloadValueSet and isExistingValueSet.

See ManageValueSetsSample class for sample code that demonstrates the attributes and operations described in this section.

getValueSets

Returns collection of ValueSetStatus objects.

1. Invoke getValueSets:

```
List valueSets = uddiNode.getValueSets();
```

2. Cast each element to ValueSetStatus and output contents:

```
for (Iterator iter = valueSets.iterator(); iter.hasNext();) {

  ValueSetStatus valueSetStatus = (ValueSetStatus) iter.next();
  System.out.println(valueSetStatus);
}
```

getValueSetDetail

Returns ValueSetStatus object for the given value set tModel key.

1. Invoke getValueSetDetail:

```
uddiNode.getValueSetDetail(
                "uddi:uddi.org:ubr:categorization:naics:2002");
```

2. Retrieve and display details:

```
String name = valueSetStatus.getName();
String displayName = valueSetStatus.getDisplayName();
boolean supported = valueSetStatus.isSupported();

System.out.println("name: " + name);
System.out.println("display name: " + displayName);
System.out.println("supported: " + supported);
```

3. Display value set properties:

```
List properties = valueSetStatus.getProperties();

for (Iterator iter = properties.iterator(); iter.hasNext();) {

  ValueSetProperty property = (ValueSetProperty) iter.next();
  System.out.println(property);
}
```

getValueSetProperty

Returns a property of a value set as a ValueSetProperty object. This is mainly for use by the administrative console to render properties of a value set as a row in a table. For example, one such property is the keyedReference which indicates whether the value set is checked.

1. Invoke getValueSetProperty:

```
uddiNode.getValueSetProperty(
                "uddi:uddi.org:ubr:categorization:naics:2002",
                ValueSetPropertyConstants.VS_CHECKED);
```

2. Read and display boolean value of the property:

```
boolean checked = valueSetProperty.getBooleanValue();

System.out.println("checked: " + checked);
```

updateValueSet

Updates value set status. Only the supported attribute can be updated (all other setter methods are used by the UDDI application).

1. Create a ValueSetStatus object specifying the tModel key and the updated supported value:

```
ValueSetStatus updatedStatus = new ValueSetStatus();
updatedStatus.setTModelKey(
               "uddi:uddi.org:ubr:categorization:naics:2002");
updatedStatus.setSupported(true);
```

2. Invoke updateValueSet:

```
uddiNode.updateValueSet(updatedStatus);
```

updateValueSets

Updates value set status for multiple value sets. As for the updateValueSet operation, only the supported attribute is updated.

1. Populate List with updated ValueSetStatus objects:

```
List valueSets = new ArrayList();

ValueSetStatus valueSetStatus = new ValueSetStatus();
valueSetStatus.setTModelKey(
              "uddi:uddi.org:ubr:categorization:naics:2002");
valueSetStatus.setSupported(false);
valueSets.add(valueSetStatus);

valueSetStatus = new ValueSetStatus();
 valueSetStatus.setTModelKey(
               "uddi:uddi.org:ubr:categorizationgroup:wgs84");
valueSetStatus.setSupported(false);
valueSets.add(valueSetStatus);

valueSetStatus = new ValueSetStatus();
valueSetStatus.setTModelKey(
               "uddi:uddi.org:ubr:identifier:iso6523:icd");
valueSetStatus.setSupported(false);
valueSets.add(valueSetStatus);
```

2. Invoke updateValueSets:

```
uddiNode.updateValueSets(valueSets);
```

loadValueSet

Loads values for a value set from a UDDI registry V3/V2 taxonomy data file on the local file system. Note: there is also a loadValueSet operation that takes a ValueSetData object but this is only for use by the user defined value set tool.

1. Invoke loadValueSet:

```
uddiNode.loadValueSet(
                     "C:/valuesets/myvalueset.txt",
                     "uddi:cell:node:server:myValueSet");
```

changeValueSetTModelKey

Any value set values that were allocated to one value set tModel are allocated to the new value set tModel.

1. Invoke changeValueSetTModelKey with old and new tModel keys:

```
uddiNode.changeValueSetTModelKey(
             "uddi:cell:node:server:myValueSet",
             "uddi:cell:node:server:myNewValueSet");
```

unloadValueSet

Unloads values for a value set with the given tModel key.

1. Invoke unloadValueSet:

```
uddiNode.unloadValueSet("uddi:myValueSet");
```

isExistingValueSet

Determines if value set data exists for the given tModel key.

1. Invoke isExistingValueSet and display result:

```
boolean exists = uddiNode.isExistingValueSet(
                 "uddi:uddi.org:ubr:categorization:naics:2002");
System.out.println("NAICS 2002 is a value set: " + exists);
```

User-defined value set support in the UDDI registry

In UDDI Version 2 this was called 'Custom Taxonomy Support'.

Data is worthless if it is lost within a mass of other data and cannot be distinguished or discovered. If a client of UDDI cannot effectively find information within a registry, the purpose of UDDI is considerably compromised. Providing the structure and modeling tools to address this problem is at the heart of UDDI's design. The verification of data within UDDI is core to its mission of description, discovery and integration. It achieves this by several means.

It allows users to define multiple value sets that can be used in UDDI. In such a way, multiple classification schemes can be overlaid on a single UDDI entity. This capability allows organizations to extend the set of such systems UDDI registries support. One is not tied to a single system, but can rather employ several different classification systems simultaneously.

While default value sets are shipped with the product, the UDDI Version 3 registry provides tools enabling 'custom' value sets to be added, potentially enabling UDDI entities to be more specifically categorized when published and further enhancing the capability of client to find specific data.

These value sets can be either checked or unchecked, and this is indicated via a keyedReference in the categoryBag of the tModel that represents a value set (a ″categorization tModel″). These keyedReferences have the tModel key for uddi-org:types and are added to the categoryBag to further describe the behavior of the categorization tModel, as follows:

checked
: Marking a tModel with this classification asserts that it represents a categorization, identifier, or namespace tModel that has a validation service to check that category values are present in a specified value set.

unchecked
: Marking a tModel with this classification asserts that it represents a categorization, identifier, or namespace tModel that does not have a validation service.

The procedure defined below describes how to add additional user-defined value sets, and display their allowed values in the UDDI user console value set tree display. Rational Application Developer has a Web

Services Explorer user interface that also allows addition and display of custom checked value sets. The publisher of a value set categorization tModel may specify a 'display name' for use in UDDI user console implementations.

Procedure for adding a user defined value set

To add a user defined value set to the UDDI registry, perform the following tasks:

1. Publish a categorization tModel
2. Load the user defined value set data
3. Set the value set to **supported** status using the Administrative console. To do this you must be a user in an administrative role. This means that user defined value sets cannot be added to the UDDI registry without administrator permission.

The checked value set will only be referenced when the above tasks are complete. Value set data must be provided for validating checked value sets.

Value set data may also be used by user consoles for unchecked value sets, but it is not a requirement and is usually only used for presentation of deprecated value sets, such as unspc-org:unspc and back-level compatibility.

If the value set is checked, any publish requests that have a categoryBag containing keyedReferences with the new categorization tModel will be validated. If there is value set data corresponding to the categorization tModel in the registry database, only valid values will be accepted. If there is no value set data in the database **all** values will be rejected, and the publish request will fail. If the categorization tModel is unchecked, all values will be allowed, regardless of whether there is a corresponding value set present in the UDDI registry database. The value set tModel is not available for use until the administrator enables support for it using the administrative console, or the JMX interface.

Suggested approach

To introduce a new value set:

1. Publish the categorization tModel with a keyedReference of type 'uddi-org:categorization:types' with a key value of **categorization**, a keyedReference of type 'uddi-org:categorization:types' with a Key Name of **'Checked value set'** and a Key Value of **'checked'**, or a Key Name of **'Unchecked value set'** and a Key Value of **'unchecked'** and a keyedReference of type 'uddi-org:categorization:general_keywords' supplying the value set display name (as described below).
2. Load user defined value set data into the UDDI registry database using the UDDIUserDefinedValueSet utility (described below).
3. Use the administrative Console to set the status of the value set to supported (as described in Value set settings). This can also be achieved directly using the JMX interface.

Note: The SOAP and EJB interfaces will be able to make use of categorization tModels as soon as they are published. However, the UDDI registry user console will require a restart of the UDDI application because it currently gathers its list of categorizations for use in the value set tree display when the application starts.

Publishing a Checked Categorization tModel

This section describes how to publish a checked categorization tModel with the **'Checked value set'** Key Name for use by a user defined value set.

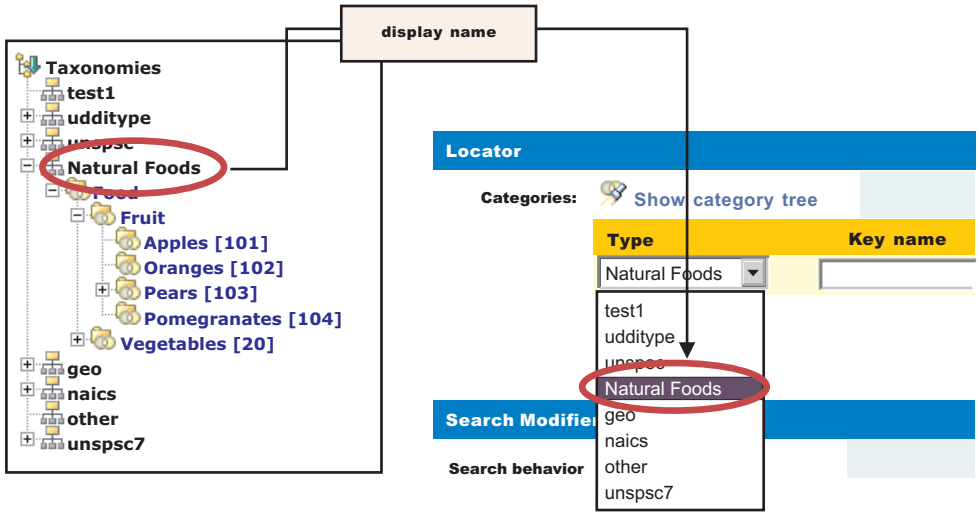
Publish a tModel to the UDDI registry with a categoryBag containing keyedReferences as follows:

Note	tModelKey	KeyName	KeyValue

1	uddi:uddi-org:categorization:types In the UDDI registry user interface this tModelKey can be chosen by selecting the category type of **UDDI Types**	**categorization**	**categorization**
2	uddi:uddi-org:categorization:types In the UDDI registry user interface this tModelKey can be chosen by selecting the category type of **UDDI Types**	**Checked value set**	**checked**
3	uddi:uddi-org:categorization:general_keywords In the UDDI registry user interface this tModelKey can be chosen by selecting the category type of **categorization:general_keywords**	**urn:x-ibm:uddi:customTaxonomy:displayName**	*<User Defined Value Set displayName>*

1. Indicates this tModel is a categorization tModel (required).
2. Indicates use of the tModel will be checked against a list of valid data (required). (Omitting this keyedReference, or explicitly specifying a value of 'unchecked' will indicate this categorization is unchecked).
3. Indicates special use of the general keywords value set, with a proprietary uniform resource name (URN) as the keyName value, defines a name for the user-defined value set that is intended for use in user console implementations where the full tModel name might be too long. The value can be 1-255 characters (inclusive) long.

The displayName is intended to provide a way to label a value set such that, when the UDDI user console displays it in a value set tree or in a pull-down list of available value sets, the meaning is clear to the user without being restricted to 8 characters and without needing to be the same as the published tModelName, which could be as long as 255 characters. An example is shown below:



The urn:x-ibm:customTaxonomy:displayName should be unique if only to avoid confusion when displayed in user interfaces but this is not validated.

To publish a new categorization tModel using SOAP, the message would be:

```
<save_tModel generic="3.0" xmlns="urn:uddi-org:api_v3">
  <authInfo></authInfo>>
  <tModel tModelKey="">
    <name>Natural Foods tModel</name>
    <categoryBag>
      <keyedReference tModelKey="uddi:uddi.org:categorization:types" keyName="categorization"
                      keyValue="categorization"/>
      <keyedReference tModelKey="uddi:uddi.org:categorization:types" keyName="Checked value set"
                      keyValue="checked"/>
      <keyedReference tModelKey="uddi:uddi.org:categorization:general_keywords"
                      keyName="urn:x-ibm:uddi:customTaxonomy:displayName" keyValue="Natural Foods"/>
    </categoryBag>
  </tModel>
</save_tModel>
```

Note: to specify an unchecked categorization substitute the key name **'Checked value set'** with **'Unchecked value set'** and **'checked'** Key Value with **'unchecked'** or, more simply, omit the keyedReference completely.

Loading User Defined Value Set Data

User Defined Value Set Data File Format

Value set data is identified by a unique code value, an optional description and a parent code that specifies its relationship with other code values. Value set data must adhere to this format:

Column name	Maximum length	Description of use
Code	765	Unique value within the value set used for validation
description	765	Typically used by UDDI user consoles and optionally in the keyedReference as the keyName value
parentcode	765	Indicates which existing **code** is the logical parent of this one, and is used in tree displays

Typically columns are delimited in the value set data file by '#' characters as in this example:

```
00#Food#00
10#Fruit#00
101#Apples#10
102#Oranges#10
103#Pears#10
1031#Anjou#103
1032#Conference#103
1033#Bosc#103
104#Pomegranates#10
20#Vegetables#00
201#Carrots#20
202#Potatoes#20
203#Peas#20
204#Sprouts#20
```

In the example, 'Food' is the description for the root node with child nodes of 'Fruit' and 'Vegetables' (both of these have parentcode values the same as the code value for 'Food').

The value set data in the example file could then be rendered in a tree like this:

```
Food
  Fruit
    Apples
    Oranges
    Pears
      Anjou
```

```
      Conference
      Bosc
    Pomegranates
  Vegetables
    Carrots
    Potatoes
    Peas
    Sprouts
```

The file must be saved in UTF-8 format.

Custom Taxonomy files used in UDDI Version 2 are also supported by the utility.

UDDIUserDefinedValueSet

A utility is provided to load value set data into the UDDI registry, assign existing value set data to another tModel and unload existing value set data. This utility uses the UDDI registry's JMX interface and therefore requires a number of connection parameters.

```
Usage: UDDIUserDefinedValueSet[.sh|.bat] '{'function'}' [options]

function:
-load <path> <key>         Load value set data from specified file
-newKey <oldKey> <newKey>  Move value set to a new tModel
-unload <key>              Unload existing value set

options:
-properties <path>         Specify location of configuration file
-host <host name>          Application Server or Deployment Manager host
-port <port>               SOAP Lister port number
-node <node name>          Node running a UDDI server
-server <server name>      Server with UDDI deployed
-columnDelimiter <delim>   Character delimiter to denote field end
-stringDelimiter <delim>   Character delimiter to denote strings


Connector security parameters

-userName <name>
-password <password>
-trustStore <path>
-trustStorePassword <password>
-keyStore <name>
-keyStorePassword <password>
```

Note: Ensure that the command window from which the UDDIUserDefinedValueSet is run is using a suitable codepage and font for displaying the characters contained in the value set name. Use of an incorrect codepage/font may result in unclear messages on a successful load, and create difficulty using the -unload and -newKey options.

The UDDIUserDefinedValueSet script is located in the *app_server_root*/bin directory.

If no connection parameters are supplied a connection is sought on the local host using firstly the Deployment Managers default SOAP port number, and, if there is no Deployment Manager running, the default Application Server SOAP port number.

Command arguments are case insensitive.

Usage examples

Load a value set data for a tModel on the local UDDI registry using the percent sign as a column marker in the valuesetdata.txt file.

```
UDDIUserDefinedValueSet.xxx -load valuesetdata.txt uddi:a708b8a7-35b5-451c-aafc-718ae071fcfe -columnDelimi
ter %
```

where .xxx is .bat for Windows operating systems or .sh for UNIX and Linux platforms.

Move value set data from one checked tModel to another on a UDDI registry in a network deployment configuration.

```
UDDIUserDefinedValueSet.xxx –newKey uddi:a708b8a7-35b5-451c-aafc-718ae071fcfe uddi:b819c9b8-46c6-562d-bb
0d-829bf1820d0f –host depmanagerhost.ibm.com –port 8879 –node uddinode –server uddiserver -override
```

where .xxx is .bat for Windows operating systems or .sh for UNIX and Linux platforms.

Unload a value set from a tModel from a server with security turned on supplying the connection and security parameters in myproperties.properties file, but supplying the server and password arguments on the command line (which augment or override those contained in the properties file).

```
UDDIUserDefinedValueSet.xxx –unload uddi:b819c9b8-46c6-562d-bb0d-829bf1820d0f –server uddiserver
–properties myproperties.properties –password myrealpassword
```

where .xxx is .bat for Windows operating systems or .sh for UNIX and Linux platforms.

The configuration file, if specified by the optional **-properties** parameter, determines a number of optional parameters. These parameters can be specified on the command line and, if so, override the values in the properties file. These parameters are largely JMX connection parameters and security parameters.

The string.delimiter is typically used where a description value contains the same character as the column delimiter character. For example, if the column.delimiter was set to ',' (a comma), and there was a value set description value of 'Fruits, citrus', you could include this in the value set data file by setting the string.delimiter to ″ (double quote) and enclosing the description in quotes: 'Fruits, citrus'. Note that the quote character is escaped with a backslash ('\') to indicate the literal character is to be used.

If an attempt is made to load a value set to a tModel that has existing value set data, a warning message is given. To override this error provide the **-override** argument. This argument is also required if moving value set data to a new tModel using **-newKey** where the tModel is **checked**, and also unloaded value set data for a **checked** tModel.

Command line arguments and example data	Property and example data	Comments
-columnDelimiter #	column.delimiter=#	Column delimiter used in value set data files
-stringDelimiter \″	string.delimiter=\″	Field delimiter (must be different to the column.delimiter value
-host ibm.com	host=ibm.com	Host name of the system running Deployment Manager or Application Server
-port 8880	port=8880	SOAP port number of Deployment Manager or Application Server
-node ibmNode	node=ibmNode	Name of the Node running the server with the UDDI registry
-server server1	server=server1	Server running the UDDI registry
-userName ibmuser	security.username=ibmuser	User name. Required if WebSphere security is turned on
-password mypassword	security.password=mypassword	Password
-trustStore /TrustStoreLocation	security.truststore=/TrustStoreLocation	Truststore file location

-keyStore ibmkeystore	security.keystore=ibmkeystore	Keystore name
-trustStorepassword trustpass	security.truststore.password=trustpass	Truststore password
-keyStorePassword keypass	security.keystore.password=keypass	Keystore password

Set the value set to supported

Use the administrative console to set the value set to **supported** by:

- Click *UDDI Nodes* > *<node>* and *Value Sets* (under Additional Properties on the right of the screen)
- Select the Value Set (by checking the box next to it)
- Click *Enable Support* above the list of Value Sets

Validation and Error Handling

The UDDI registry user console performs validation while a save tModel request is being built, that is, before the publish occurs. For example, if the user tries to add two customTaxonomy:displayName keyedReferences the following message is displayed:

```
Advice: Only one 'urn:x-ibm:uddi:customTaxonomy:displayName' key name is allowed for the 'Other' taxonomy.
```

If a keyedReference containing a keyName value that starts with 'urn:x-ibm:uddi:customTaxonomy:' is followed by anything other than 'displayName', the following message is displayed:

```
Advice: Only key name values of 'urn:x-ibm:uddi:customTaxonomy:displayName' are supported.
```

For requests where the save_tModel message may have multiple tModels, if any one of the tModels is a categorization tModel and it fails validation, the request fails with a UDDIInvalidValueException (plus additional information explaining the likely cause), and none of the tModels is published. For example:

```
E_invalidValue (20200) A value that was passed in a keyValue attribute did not pass validation.
This applies to checked categorizations, identifiers and other validated code lists. The error
text will clearly indicate the key and value combination that failed validation. Invalid 'custo
mTaxonomy:dbKey' keyValue [naics] in keyedReference. KeyValue already in use by tModelKey[UUID:
C0B9FE13-179F-413D-8A5B-5004DB8E5BB2]
```

UDDI Utility Tools

The UDDI Utility Tools is a suite of functions that can be used to migrate, move or copy UDDI Version 2 entities, including child entities and their respective Version 2 entity keys, into a Version 3 UDDI registry.

Note: The UDDI Version 3 publish API supports publisher assigned keys (the Version 2 API did not) and promotion of entities between Version 3 registries can be achieved using normal API functions. UDDI Utility Tools supplied in this release is functionally equivalent to the version supplied in WebSphere Application Server 5.1. However, it is important to know that all UDDI Utility Tools functions in this release are performed using the UDDI Version 2 API. You can export from Version 2 and 3 registries (supplying only the Version 2 representation of the UDDI Entity key) and import into the Version 3 registry, using Version 2 API types. Entities from a Version 3 registry are exported as Version 2 entities and, as such, elements such as digital signatures will not be present. See section Saving Version 3 entities with a supplied key for an example on how to use the Version 3 API to assign your own keys to Version 3 entities.

Other uses of the tool include:

- Search and select entities from a source UDDI registry by specifying Version 2 keys or search criteria
- Publishing canonical tModels in a UDDI registry, including child entities
- Persist UDDI (Version 2) entities in an intermediate XML representation that can be used to customize and copy those entities to multiple target UDDI Registries, by specifying Version 2 Keys
- Update existing entities in a target UDDI registry, including child entities

- Delete selected entities from a target UDDI registry by specifying Version 2 keys

Use the UDDI Utility Tools by running the UDDIUtilityTools.jar file. This file is located in the *app_server_root*/UDDIReg/scripts directory. Alternatively, you can invoke all of the functions of UDDI Utility Tools through the supplied public Java API.

There are five main functions in UDDI Utility Tools:

Export
Given an entity type and key, or a list of entity types and keys, UDDI Utility Tools gets the UDDI entities from the specified registry and writes them to the UDDI Entity Definition File. The entity type for each key can be one of business, service, bindingTemplate or tModel. The Entity Definition File contains XML that exactly describes each of the specified entities, according to the UDDI Utility Tools schema (which includes the UDDI Version 2 schema). The UDDI Entity Definition File separates entities by type, and automatically detects and records tModels referenced by the specified entities. You can use the 'referenced tModels' section of the file to ensure a target registry includes any referenced tModels before you try to import new entities to that registry.

Import
Given a list of UDDI entities (which can be supplied using the UDDI Entity Definition File generated by the export function, possibly with additional editing, or programmatically in a container object), the import function detects if the entities already exist in the target registry and, if they do not, creates a minimal entity (″stub″) with the specified key. The entities are then published updating the stubs with the supplied data and overwriting, or ignoring, existing entities as specified by the user. Note that the original key is maintained throughout.

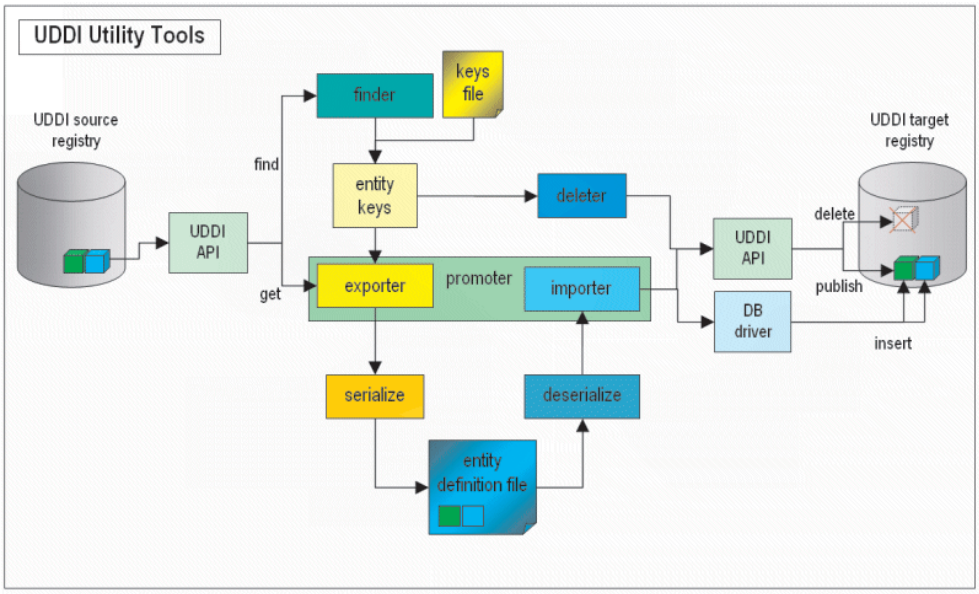
Promote
Combines the export and import steps such that the specified entities are extracted (by key) from the source registry and then imported into the target registry in a single logical step. The generation of a UDDI Entity Definition File is optional for this function.

Delete Deletes the specified entities from the target UDDI registry. The entities to delete are specified as an entity type, or a list of entity types, and keys, in the same way as for the export function.

Find Matching Entities
Takes as input search criteria in the form of UDDI Inquiry API objects for each of the various entity types. The set of entities that match the search criteria are used to generate a list of entity keys, and this in turn can be used as input to the export, promote and delete functions.

Note: This function is available only through the programmatic API.

The relationship between the functions, their input and output, and the source and target UDDI Registries is shown in this conceptual overview diagram:



Setting up the configuration file

Configuration data for UDDI Utility Tools resides in a configuration properties file, which describes the runtime environment, UDDI and database locations and access information, logging information, security configuration, entity definition file location, and other flags to control whether referenced entities are to be imported and/or overwritten.

UDDI Utility Tools is distributed with a sample configuration properties file (UDDIUtilityTools.properties) and this is searched for by default in the current directory if no properties path is specified. By default, this file is located in the *app_server_root*/UDDIReg/scripts directory. Modify the file, according to the following list, and specify this modified file when running the utility tools.

- Set the classpath, which should include the current directory (.) and the UDDIUtilityTools.jar itself, plus all the dependent jars, which are listed in the Prerequisites section later on in this topic. The classpath must include the database driver jar (for example db2java.zip).

  If you are configuring a JSSE provider, add the .jar file which contains the provider to the classpath. The configuration of a JSSE provider is optional and is performed by setting the `jsse.provider` property. The default value is com.ibm.jsse.IBMJSSEProvider. To specify the FIPS JSSE provider set the value of the `jsse.provider` property to com.ibm.fips.jsse.IBMJSSEFIPSProvider.
- Set other properties, which are commented in the sample UDDIUtilityTools.properties file as shown below.
- Change localhost to the name of your server.
- Change the port number 9080 to your internal HTTP port.

Note: Windows Use forward slashes in paths. Back slashes can be interpreted as escape sequences such as tab spaces. For example,

```
C:\temp\definitions\entities01.xml
```

becomes

```
C: emp\definitions\entities01.xml
```

```
###############################################
# Runtime environment                         #
# (if invoking via java -jar...)              #
# "X Y" required around paths with spaces.    #
# Replace WAS_HOME with your WAS home path.   #
# Replace DB2_HOME with the locations of DB2  #
#                                             #
# db2java.zip is for DB2 - replace this with  #
# appropriate database driver file.           #
###############################################
classpath=.;WAS_HOME/UDDIReg/scripts/UDDIUtilityTools.jar;WAS_HOME/plugins/com.ibm.ws.runtime_6.1.0.
jar;WAS_HOME/plugins/com.ibm.uddi_1.0.0.jar;WAS_HOME/lib/j2ee.jar;"DB2_HOME/SQLLIB/java/db2java.zip"

###############################################
# SOAP entry points for source UDDI           #
###############################################
fromInquiryURL=http://localhost:9080/uddisoap/inquiryapi
fromGetURL=http://localhost:9080/uddisoap/get

###############################################
# SOAP entry points for target UDDI           #
###############################################
toInquiryURL=http://localhost:9080/uddisoap/inquiryapi
toPublishURL=http://localhost:9080/uddisoap/publishapi

###############################################
# UDDI Registry user information              #
#                                             #
# Note: this must match the user information  #
# that was used to publish the entities on    #
# the target UDDI registry.                   #
###############################################
userID=UNAUTHENTICATED
password=NONE

###############################################
# Configuration for destination UDDI DB       #
###############################################
dbDriver=COM.ibm.db2.jdbc.app.DB2Driver
dbUrl=jdbc:db2:uddi30
dbUser=db2admin
dbPasswd=db2admin

###############################################
# Security provider configuration             #
###############################################
# Indicates whether security is required on the target registry
secure.connection=true

# The location of the truststore if security is required
trustStore.fileName=TrustFile.jks

# The password for the trust store
trustStore.password=WebAS

# The JSSE Provider class name
jsse.provider=com.ibm.jsse.IBMJSSEProvider

###############################################
# Trace and message logging configuration     #
###############################################
# detail level of message output (all functions)
verbose=true

# detail level of trace output.
# 1: severe
# 2: normal
```

```
# 3: detail
traceLevel=3

# path to message log file (relative or absolute)
messageLogFileName=logs/messages.log

# path to trace log file (relative or absolute)
traceLogFileName=logs/trace.log

##########################################
# Miscellaneous Options                  #
##########################################
# indicates if existing entities are overwritten (import/promote)
# Note: tModels in referencedTModels section are never overwritten,
#       regardless of this setting. To overwrite tModels, they must
#       be present in the tModels section.
overwrite=false

# indicates if referenced entities will be imported (import/promote)
importReferencedEntities=true

# location of entity definition file, used for (export/import)
UddiEntityDefinitionFile=definitions/entities01.xml

# namespace prefix to use in definition file (export)
namespacePrefix=promote
```

Prerequisites

Windows To run the UDDI Utility Tools you must use the IBM Development Kit for Java code that is supplied with WebSphere Application Server. This Development Kit is located in *app_server_root*/java/bin.

Ensure that the following .jar files are available to the UDDI Utility Tools. The locations of the .jar files should be specified in the classpath property in the UDDI Utility Tools properties file:

UDDIUtilityTools.jar
This is the tools JAR itself and is located in *app_server_root*/UDDIReg/scripts.

com.ibm.uddi_1.0.0.jar
This file contains the UDDI4J classes and is located in *app_server_root*/plugins.

j2ee.jar
This file contains some required J2EE classes and is located in *app_server_root*/lib.

com.ibm.ws.runtime_6.1.0.jar
This is the Apache SOAP implementation and is located in *app_server_root*/plugins.

DbDriver
This is the driver needed to allow the UDDIUtilityTool to connect to your target database. See the table below for the values you need to specify for your chosen database:

	DB2	Cloudscape	Oracle
DBDriverLocation for classpath	*DB2_HOME*/db2java.zip	*app_server_root*/derby/lib/derbyclient.jar	*ORACLE_HOME*/jdbc/lib/ojdbc14.jar
Driver	COM.ibm.db2.jdbc.app.DB2Driver, or com.ibm.db2.jcc.DB2Driver if you are using a remote DB2 database (you can also set up a local alias to the remote database using the DB2 client)	com.ibm.db2.jcc.DB2Driver	oracle.jdbc.OracleDriver

	DB2	Cloudscape	Oracle
URL	jdbc:db2://host:*database_name*	jdbc:db2j:net://host:1527/ *database_name* (see note below)	jdbc:oracle:thin:@host:1521: *database_name*

where

- *app_server_root* is the directory location of WebSphere Application Server.
- *DB2_HOME* is the directory location of DB2, for example c:\Program Files\SQLLIB\java12\
- *ORACLE_HOME* is the directory location of Oracle, for example c:\oracle\ora92\
- *database_name* is the name of the database. For Cloudscape, make sure that *database_name* includes the path to the database, for example *profile_root*/databases/com.ibm.uddi/UDDI30

Notes:

- If you are using Cloudscape, make the database network enabled so that it can handle multiple connections. Refer to the managing derby network server section of the Cloudscape information center for details on how to do this.
- If you are using DB2, add DB2_HOME/sqllib/lib to your LD_LIBRARY_PATH and LIBPATH environment variables.

The Security provider configuration section in the above properties file shows the location of the default DummyClientTrustFile.jks file. If you are using your own truststore, ensure that the location is placed here.

The UDDI Utility Tools use UDDI Version 2 SOAP inquiry and publish interfaces. These APIs are protected as described in Access control for UDDI registry interfaces. The UDDI Utility Tools also access the UDDI registry database through the database driver, and access to the database is controlled by the database management system.

The UDDI Entity Definition File

You generate this file by the export and promote functions, or you can choose to create it (either by hand, or by modifying a version of the file output by UDDI Utility Tools specifying the export function). It is the input to the import function.

Note: The extension to the uddi:tModel type to add a 'deleted' attribute is not currently used in UDDI Utility Tools.

The file is validated for well formedness and that it complies with the UDDI Utility Tools schema, shown here.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema id="uddiPromote" attributeFormDefault="unqualified" elementFormDefault="qualified"
 targetNamespace="http://www.ibm.com/xmlns/prod/WebSphere/UDDIUtilityTools" xmlns:xsd="http://www.w3.org
                  /2001/XMLSchema"
xmlns:uddi="urn:uddi-org:api_v2" xmlns="http://www.ibm.com/xmlns/prod/WebSphere/UDDIUtilityTools"
xmlns:promote="http://www.ibm.com/xmlns/prod/WebSphere/UDDIUtilityTools">

 <xsd:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml.xsd" />
 <xsd:import namespace="urn:uddi-org:api_v2" schemaLocation="uddi_v2.xsd" />

 <!-- define a type to represent state of a tModel -->
 <xsd:simpleType name="tModelDeleted">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="true" />
   <xsd:enumeration value="false" />
  </xsd:restriction>
 </xsd:simpleType>

 <!-- extend tModel with additional attribute of type tModelDeleted -->
 <!-- This is restricted to values true or false -->
```

```
<xsd:complexType name="tModel">
 <xsd:complexContent>
  <xsd:extension base="uddi:tModel">
   <xsd:attribute name="deleted" type="promote:tModelDeleted" use="optional" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>


<!-- Top level element definitions -->
<xsd:element name="uddiEntities" type="promote:uddiEntities" />
<xsd:complexType name="uddiEntities">
 <xsd:sequence>
  <xsd:element ref="promote:tModels" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:businesses" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:services" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:bindings" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:referencedTModels" minOccurs="0" maxOccurs="1" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="businesses" type="promote:businesses" />
<xsd:complexType name="businesses">
 <xsd:sequence>
  <xsd:element ref="uddi:businessEntity" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="tModels" type="promote:tModels" />
<xsd:complexType name="tModels">
 <xsd:sequence>
  <xsd:element ref="uddi:tModel" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="services" type="promote:services" />
<xsd:complexType name="services">
 <xsd:sequence>
  <xsd:element ref="uddi:businessService" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="bindings" type="promote:bindings" />
<xsd:complexType name="bindings">
 <xsd:sequence>
  <xsd:element ref="uddi:bindingTemplate" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="referencedTModels" type="promote:referencedTModels" />
<xsd:complexType name="referencedTModels">
 <xsd:sequence>
  <xsd:element ref="uddi:tModel" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

UDDI Entity Definition File example for canonical tModels

The example Entity Definition File following shows the five main sections for tModels, businesses, services, bindings and referencedTModels:

UDDI Utility Tools can be used to create new UDDI entities in a target UDDI registry. A typical example of this is to introduce a new canonical tModel, which has a publicly known tModel key.

```
<?xml version="1.0" encoding="UTF-8"?>
<promote:uddiEntities xmlns="urn:uddi-org:api_v2" xmlns:promote="http://www.ibm.com/xmlns/prod/WebSphere/
                      UDDIUtilityTools">

  <!-- tModels -->
  <promote:tModels>

    <tModel tModelKey="uuid:ee3966a8-faa5-416e-9772-128554343571" >
      <name>http://schemas.xmlsoap.org/ws/2002/07/policytmodel</name>
      <description>WS-PolicyAttachment policy expression</description>
    </tModel>

    <tModel tModelKey="uuid:ad61de98-4db8-31b2-a299-a2373dc97212" >
      <name>uddi-org:wsdl:address</name>
 <description xml:lang="en">
This tModel is used to specify the URL fact that the address must be obtained from the WSDL deployment
file.
      </description>
      <overviewDoc>
        <overviewURL>
http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm#Address
        </overviewURL>
      </overviewDoc>
    </tModel>

  </promote:tModels>

  <!-- businesses -->
  <promote:businesses>
  </promote:businesses>

  <!-- services -->
  <promote:services>
  </promote:services>

  <!-- bindings -->
  <promote:bindings>
  </promote:bindings>

  <!-- referenced tModels -->
  <promote:referencedTModels>
  </promote:referencedTModels>

</promote:uddiEntities>
```

Starting UDDI Utility Tools at a command prompt

Ensure that you are using the correct level of Java code by setting the PATH statement to include the Java code that is supplied with WebSphere Application Server. For example, from the command line, type:

- Windows

  ```
  set PATH=app_server_root\java\bin;%PATH%
  ```

- Linux

  ```
  export PATH=app_server_root/java/bin:$PATH
  ```

If you are using DB2 on UNIX and Linux platforms run the db2profile script before issuing the java command to start UDDI Utility Tools. This script is located within the DB2 instance home directory under sqllib and is invoked by typing:

```
. /$DB2_HOME/db2profile
```

Note: In the above example, notice that the '.' is followed by a single space character.

Note: On UNIX and Linux platforms the DB2 user **must** have a db2profile at $HOME/sqllib/db2profile.

UDDI Utility Tools can be started using:

java - jar UDDIUtilityTools.jar <function> [options]
: using a specified properties file that sets up classpath and other parameters, or it can be called using:

java CommandLineProcessor
: where CommandLineProcessor is the class which processes command line arguments for UDDI Utility Tools, sets up configuration and invokes the appropriate function.

Note: Before executing UDDIUtilityTools.jar from the command line, ensure that you have edited the UDDIUtilityTools.properties file. If you have saved this properties file in a different directory from the directory containing the UDDIUtilityTools.jar file, make sure you specify the location of the properties file as part of the command line arguments. See the Setting up the configuration file section earlier in this topic for more details.

The usage is as follows:

```
Usage: java -jar UDDIUtilityTools.jar {function} [options]

function:
  -promote <entity source>   Promote entities between registries
  -export <entity source>    Extract entities from registry to XML
  -delete <entity source>    Delete entities from registry
  -import                    Create entities from XML to registry

where <entity source> is one of:
  -tmodel|-business|-service|-binding <key> Specify single entity type and key
  -keysFile | -f <filename>  Specify file containing entity types and keys

options:
  -properties <filename>     Specify path to configuration file
  -overwrite | -o            Overwrite an entity if it already exists
  -log | -v                  Output verbose messages
  -definitionFile <filename> Specify path to UDDI entity definition file
  -importReferenced          Import entities referenced by source entities

The following options override property settings in configuration file:
  -overwrite
  -log
  -definitionFile
  -importReferenced

Example: java -jar UDDIUtilityTools.jar -promote -keysFile C:/uddikeys.txt
```

Below are a set of UDDI Utility Tools command line examples. The examples use the Windows operating systems file system:

To export a single business to the EDF file specified in a properties file in the current directory.

```
java -jar UDDIUtilityTools.jar -export -business 28B8B928-2B2E-4EC9-A647-1E40651E4752
```

As above but this time using a keys file to specify the entities to be exported

```
java -jar UDDIUtilityTools.jar -export -keysFile C:/myKeyFiles/keyFile01.txt
```

As above but also specifying verbose output to appear on the command line.

```
java -jar UDDIUtilityTools.jar -export -keysFile C:/myKeyFiles/keyFile02.txt -v
```

To import the contents of the default EDF specified in a UDDIUtilitiyTools.properties file in the current directory.

```
java -jar UDDIUtilityTools.jar -import
```

As above but also specifying that referenced tModels should be imported into the target registry.

```
java -jar UDDIUtilityTools.jar -import -importReferenced
```

To import the entities from an EDF at the specified location. Note the use of forward slashes even though this is an example on a Windows operating systems file system.

```
java -jar UDDIUtilityTools.jar -import -definitionFile C:/myEDFs/entities01.xml
```

To import the entities from the default EDF including referenced tModels. Overwrite specifies that any entities excluding referenced tModels that are found in the target registry should be overwritten.

```
java -jar UDDIUtilityTools.jar -import -overwrite -importReferenced
```

To promote a single service from a source to a target registry using the properties file at a specified location.

```
java -jar UDDIUtilityTools.jar -promote -service 67961D67-330F-4F14-8210-E74A58E710F3
-properties C:/UUT/myUUTProps.properties
```

To promote a set of entities specified in a keys file.

```
java -jar UDDIUtilityTools.jar -promote -keysFile C:/myKeyFiles/keyFile03.txt
```

As above but specifying that existing entities in the target registry get overwritten.

```
java -jar UDDIUtilityTools.jar -promote -keysFile C:/myKeyFiles/keyFile04.txt -overwrite
```

To promote a set of entities specified in a keys file including referenced tModels.

```
java -jar UDDIUtilityTools.jar -promote -keysFile C:/myKeyFiles/keyFile05.txt -importReferenced
```

To promote a set of entities specified in a keys file but also create an EDF containing the promoted entities.

```
java -jar UDDIUtilityTools.jar -promote -keysFile C:/myKeyFiles/keyFile06.txt
-definitionFile C:/myEDFs/entities02.xml
```

To logically delete a single tModel. Note that it is not possible to physically delete tModels.

```
java -jar UDDIUtilityTools.jar -delete -tModel UUID:1E2B9D1E-E53D-4D36-9D46-6CCC176C466A
```

To delete all the entities specified in the keys file. Note that with the exception tModels all other entities will be physically deleted from the target registry.

```
java -jar UDDIUtilityTools.jar -delete -keysFile C:/myKeyFiles/keyFile04.txt
```

A keys file example

Below is an example of the keys that are to be exported, promoted or deleted from the target registry:

```
#
# Keys of entities to be exported, promoted from source registry or deleted from target registry
#
# Note: keys must be comma separated and on SAME line
# Note: property names are case sensitive. ('tmodels=' will be ignored)

businesses=97C77097-AC6C-4CA0-A6C4-452F7045C470, 4975E949-581F-4FCA-AD5F-E08280E05F9F
services=BB3864BB-1578-4833-8179-14391F14791F
bindings=
tModels=273F1727-7BFF-4FB5-A1FD-BA5C45BAFD9C
```

Note: If the importReferenced property is set to true, the list of tModels in the referencedTModels section is imported to the target registry. Minimal entities are created if the referencedTModel is new. If the referencedTModel already exists it is never overwritten, regardless of the overwrite property value. This is so that commonly referenced tModels such as categorization tModels do not keep being updated unnecessarily.

Should you need to update a referencedTModel, you must manually move the referencedTModel definition to the tModels section in the entity definition file and set overwrite to true.

Content of the log files

Below shows examples of contents of two of the log files produced by running the tool. Note that some comments have been added in square brackets and in italic to highlight important points within the log file. The first is the messages.log which shows successful and unsuccessful operations for export, import and delete functions:

```
[29/07/04 17:39:57:531 BST] CWUDU0002I: ********** Starting UDDI Utility Tools ********** [timestamp and
 eyecatcher indicate when tool is run]
[29/07/04 17:39:57:531 BST] CWUDU0009I: Exporting entities...
[29/07/04 17:39:57:531 BST] CWUDU0015I: Exported 14 entities.
[29/07/04 17:39:57:531 BST] CWUDU0029I: Serializing...
[29/07/04 17:39:57:531 BST] CWUDU0030I: Serialized entities.
[29/07/04 17:39:57:531 BST] CWUDU0016I: Importing entities...
[29/07/04 17:39:57:531 BST] CWUDU0124I: Created tModel minimal entity with tModelKey [uuid:667e2766-4781-
4151-b3a0-809f7180a096].
[29/07/04 17:39:57:531 BST] CWUDU0121I: Created business minimal entity with businessKey [263f5526-8708-4
834-9f5d-8f8c878f5d6e].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [0af2a30a-be70-401
f-a027-331a6c332712].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [61012761-d02c-4c7
0-ae98-435ffd4398f9].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [f97af9f9
-7cb7-47bd-8b90-b55e4db590df].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [17e4c017
-d273-43ec-af4a-f9b841f94a30].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [9e2c239e
-3b30-40a9-9c25-ce64edce25b9].
[29/07/04 17:39:57:531 BST] CWUDU0121I: Created business minimal entity with businessKey [49bb6949-4b0e-4
e81-88a7-e26bfbe2a7f1].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [003d2b00-f6c0-407
1-8b84-f235a2f28445].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [df1019df
-2d2f-4f32-bf18-4f21274f1835].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [b229aeb2
-f2b1-4115-a06f-536753536f10].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [84d8e584-2510-409
9-9b2a-6023f1602a0a].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [62a9a762
-7fff-4f7a-8463-af0c79af63ee].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [e08654e0
-b212-42c0-bcf3-655e9765f392].
[29/07/04 17:39:57:531 BST] CWUDU0115I: Imported 7 entities and 0 referenced entities. [this kind of
message indicates the operation worked!]
[29/07/04 17:39:57:531 BST] CWUDU0002I: ********** Starting UDDI Utility Tools **********
[29/07/04 17:39:57:531 BST] CWUDU0023I: Deleting entities...
[29/07/04 17:39:57:531 BST] CWUDU0028I: Deleted 7 entities.
```

The second log file shows a typical trace log file entry for an export:

```
[29/07/04 17:39:57:531 BST] ********** Starting UDDI Utility Tools ********** [eyecatcher and timestamp
 indicate when tool is run]
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()  [the '>' indicates
entry to the constructor of this class]
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.export.KeyFileReader()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded tModel keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded business keys
TransformConfiguration:
  nameSpacePrefix=promote
  uddiEntityDefinitionFile=C:\temp/MigToolFiles/Results/Promote_api_EDF_1.xml

ExportConfiguration:
  fromGetURL=http://yottskry:9080/uddisoap/
```

```
  fromInquiryURL=http://yottskry:9080/uddisoap/inquiryAPI

ImportConfiguration:
  overwrite=true
  uddiEntityDefinitionFile=C:\temp/MigToolFiles/Results/Promote_api_EDF_1.xml
  importReferencedEntities=true

PublishConfiguration:
  toInquiryURL=http://davep:9080/uddisoap/inquiryAPI
  toPublishURL=http://yottskry:9080/uddisoap/publishAPI
  userID=Publisher1
  trustStoreFileName=C:\WebSphere600/AppServer/etc/DummyClientTrustFile.jks
  secureConnection=false

DatabaseConfiguration:
  dbDriver=COM.ibm.db2.jcc.DB2Driver
  dbURL=jdbc:db2:LOC1
  dbUser=db2admin

LoggerConfiguration:
  messageStream=null
  messageLogFileName=C:\temp/MigToolFiles/logs/message.log
  traceLogFileName=C:\temp/MigToolFiles/logs/trace.log
  traceLevel=3
  verbose=true


[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.PromoterAPI()
[29/07/04 17:39:57:531 BST] ********** Starting UDDI Utility Tools **********
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.export.KeyFileReader()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded tModel keys [ log
entries without a '>' or '<' are status messages only ]
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded business keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded service keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded binding keys
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.UddiEntityKeys()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.UddiEntityKeys() [the '<' indicates exit from the
constructor]
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() removed duplicate, empty
and null keys
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.export.KeyFileReader()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.deleteEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.publish.EntityDeleter()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.publish.EntityDeleter()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.UDDIClient()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.UDDIClient() client type: 1
```

Starting UDDI Utility Tools through the API

UDDI Utility Tools provides a public API to functions for exporting, importing, promoting, finding and deleting UDDI entities. All of these functions can be invoked by using the PromoterAPI class. Usage of this class to perform these functions is typically to:

1. Create a Configuration object and populate it from a Properties object or from a configuration properties file.
2. Create a PromoterAPI object passing the Configuration in the constructor.
3. For keys based functions (export, delete and promote), set the keys by supplying a UDDIEntityKeys object, the location of the keys file, or, for one entity, by specifying an entity type and a key value.
4. Invoke the corresponding method for the function required: exportEntities, promoteEntities(boolean), importEntities, deleteEntities or extractKeysFromInquiry(FindTModel, FindBusiness, FindService, FindBinding, FindRelatedBusinesses).

There is some sample code for UDDI Utility Tools, demonstrating usage of the API classes, available from Samples Central.

The ″low-level″ API classes and methods have been deprecated in this release. Refer to the API documentation for details.

Known limitations with UDDI Utility Tools and workarounds

There are some known limitations with UDDI Utility Tools and a workaround for each. See UDDI troubleshooting tips for more information.

Embedded Cloudscape Restriction

The 'export' and 'delete' functions when referencing a source registry with an embedded Cloudscape database are supported. However, the 'import' and 'promote' functions are not supported when referencing a target registry because of a limitation with the UDDI registry when working with an embedded Cloudscape database. To allow the 'promote' and 'import' functions to work, the embedded Cloudscape database needs to be made network enabled. For information about configuring network Cloudscape, refer to the managing derby network server section of the Cloudscape information center.

Saving Version 3 entities with a supplied key

An example of saving a Version 3 business with a defined key is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <save_business xmlns="urn:uddi-org:api_v3">
      <authInfo>a399c4a3-6387-47cd-a1bd-91f7bb91bdd7</authInfo>
      <businessEntity businessKey="uddi:mycompany-p1.com:computers">
        <name xml:lang="en">WithKey</name>
      </businessEntity>
    </save_business>
  </Body>
</Envelope>
```

Known limitations with UDDI Utility Tools and workarounds

There are known limitations with the UDDI Utility Tools and a workaround for each:

- PublisherAssertions are not supported and will not be promoted.

  Workaround: After the user has promoted the businesses that are related, he must recreate the publisherAssertion relationship.
- Referenced businesses in service projections are not added automatically to the EDF in the same manner as referenced tModels.

  Workaround: Add the referenced business that will 'own' the projected service to the EDF. If the business is not present in the target registry, it should be placed before the service's owning business in the EDF.
- Cycle detection for service projections are not detected in the same manner as for referenced tModels.

  Workaround: If a circular reference is present between two or more service projections, break the cycle by removing one of the projections temporarily, perform the import and update the changed entity to reestablish the cycle in the target registry.
- tModels that were deleted (in the logical sense) in the source registry are imported and promoted as undeleted in the target registry. This is because, in the UDDI Version 2 specification, the deleted state of tModels is not exposed as API calls.

  Workaround: After importing the tModel, perform a delete. This is done using the UDDI Utility Tools delete function, or any other UDDI registry API access method.

- BindingTemplates referenced by hostingRedirectors are not added automatically to the EDF in the same manner as referenced tModels.

  Workaround: Add the referenced bindingTemplate to the EDF.
- Businesses referenced by an 'owningBusiness' keyedReference are not automatically added to the EDF.

  Workaround: Import the referenced business into the target registry before importing the tModel that references it.
- A few combinations of command line arguments are not validated and prevented, for example, it is possible to specify -import with -keysFile <path to file> in the same command, although the -keysFile is ignored.

Java API for XML Registries (JAXR) Provider for UDDI

Overview

The Java API for XML Registries (JAXR) is a Java client API for accessing both UDDI (Version 2 only) and ebXML registries. It is part of the J2EE 1.4 specification.

The JAXR API comprises the J2EE packages javax.xml.registry and javax.xml.registry.infomodel. J2EE API documentation can be found at http://java.sun.com/webservices/reference/api/index.html (this is a download site). More information on JAXR, including the JAXR Version 1.0 specification, can be found at http://java.sun.com/xml/jaxr/index.jsp.

Note that the preferred UDDI Java client APIs are 'UDDI4J Version 2' for UDDI Version 2, and the 'UDDI Version 3 Client for Java' for UDDI Version 3.

JAXR Provider

The current JAXR specification (Version 1.0) defines a JAXR Provider as an implementation of the JAXR API. A JAXR Provider may be a JAXR Provider for UDDI, a JAXR Provider for ebXML, or a pluggable provider which supports both UDDI and ebXML. The JAXR Provider for UDDI is a provider for UDDI only.

UDDI Versions

A JAXR Provider for UDDI accesses a UDDI registry using the UDDI Version 2 SOAP APIs only. The UDDI registry for UDDI Version 3 in this version of WebSphere Application Server supports the UDDI Version 1, 2 and 3 SOAP APIs, and so the JAXR Provider for UDDI can be used to access this registry. The IBM JAXR Provider can also be used to access the UDDI registry for UDDI Version 2 in WebSphere Application Server Version 5.x. Note that to use the UDDI Version 3 SOAP APIs, JAXR cannot be used. The UDDI Version 3 Client for Java is recommended for this.

Capability Level

The JAXR specification defines two Capability Profiles, Capability Level 0 and Capability Level 1. The JAXR API documentation categorizes each JAXR method as either Level 0 or Level 1. A JAXR provider for UDDI has Capability Level 0 and supports all Level 0 methods. A JAXR provider for ebXML has Capability Level 1 and supports all Level 0 and Level 1 methods. The JAXR Provider for UDDI is a Capability Level 0 Provider, and supports only Level 0 methods.

JAXR for UDDI - getting started and advanced topics

Getting started

A simple sample

The following sample program shows how to obtain the ConnectionFactory instance, create a Connection to the registry and save an Organization in the registry.

```
import java.net.PasswordAuthentication;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.Properties;
import java.util.Set;

import javax.xml.registry.BulkResponse;
import javax.xml.registry.BusinessLifeCycleManager;
import javax.xml.registry.Connection;
import javax.xml.registry.ConnectionFactory;
import javax.xml.registry.JAXRException;
import javax.xml.registry.RegistryService;
import javax.xml.registry.infomodel.Key;
import javax.xml.registry.infomodel.Organization;

public class JAXRSample
{
    public static void main(String[] args) throws JAXRException
    {
        //Tell the ConnectionFactory to use the JAXR Provider for UDDI
        System.setProperty("javax.xml.registry.ConnectionFactoryClass",
                                                "com.ibm.xml.registry.uddi.ConnectionFactoryImpl");
        ConnectionFactory connectionFactory = ConnectionFactory.newInstance();

        //Set the URLs for the UDDI inquiry and publish APIs.
        //These must be the URLs of the UDDI version 2 APIs.
        Properties props = new Properties();
        props.setProperty("javax.xml.registry.queryManagerURL", "http://localhost:9080/uddisoap/inquiryapi");
        props.setProperty("javax.xml.registry.lifeCycleManagerURL", "http://localhost:9080/uddisoap/publishapi");
        connectionFactory.setProperties(props);

        //Create a Connection to the UDDI registry accessible at the above URLs.
        Connection connection = connectionFactory.createConnection();

        //Set the user ID and password used to access the UDDI registry.
        PasswordAuthentication pa = new PasswordAuthentication("Publisher1", new char[] { 'p', 'a', 's',
                                                                                  's', 'w', 'o', 'r', 'd' });
        Set credentials = new HashSet();
        credentials.add(pa);
        connection.setCredentials(credentials);

        //Get the javax.xml.registry.BusinessLifeCycleManager interface, which contains
        //methods corresponding to UDDI publish API calls.
        RegistryService registryService = connection.getRegistryService();
        BusinessLifeCycleManager lifeCycleManager = registryService.getBusinessLifeCycleManager();

        //Create an Organization (UDDI businessEntity) with name "Organization 1".
        Organization org = lifeCycleManager.createOrganization("Organization 1");

        //Add the Organization to a Collection, ready to be saved in the UDDI registry.
        Collection orgs = new ArrayList();
        orgs.add(org);

        //Save the Organization in the UDDI registry.
        BulkResponse bulkResponse = lifeCycleManager.saveOrganizations(orgs);

        //Obtain the Organization's Key (the UDDI businessEntity's businessKey) from the response.
        if (bulkResponse.getExceptions() == null)
        {
            //1 Organization was saved, so 1 key will be returned in the response collection
            Collection responses = bulkResponse.getCollection();
            Key organizationKey = (Key)responses.iterator().next();
            System.out.println("\nOrganization Key = " + organizationKey.getId());
        }
    }
}
```

Classpath

The class libraries of the JAXR Provider for UDDI are contained within the com.ibm.uddi_1.0.0.jar file, located in the *app_server_root*/plugins directory. When using the JAXR API from within a J2EE application running under WebSphere Application Server, all required classes will automatically be on the classpath. When using the JAXR API from outside this environment, the following jars must be on the Java classpath:

- *app_server_root*/lib/bootstrap.jar
- *app_server_root*/lib/j2ee.jar
- *app_server_root*/plugins/com.ibm.uddi_1.0.0.jar
- *app_server_root*/plugins/com.ibm.ws.runtime_6.1.0

javax.xml.registry.ConnectionFactory

To use the JAXR Provider for UDDI, the name of the ConnectionFactory implementation class must first be specified by setting the System Property “javax.xml.registry.ConnectionFactoryClass” to “com.ibm.xml.registry.uddi.ConnectionFactoryImpl”. Failure to specify this will result in the value defaulting to “com.sun.xml.registry,common.ConnectionFactoryImpl”, which will not be found. This will result in a JAXRException when the ConnectionFactory.newInstance() method is called. The JAXR Provider for UDDI does not support lookup of the ConnectionFactory via JNDI.

javax.xml.registry.Connection Properties

Connection specific properties must be specified by setting a java.util.Properties object on the JAXR ConnectionFactory before obtaining a Connection. The JAXR specification defines the full list of these properties. The table below lists the three most important properties, and what values they should take in order to use the JAXR Provider for UDDI to access the UDDI registry. The only required Connection property is “javax.xml.registry.queryManagerURL”, however it is recommended that “javax.xml.registry.lifeCycleManagerURL” is also set, and that the default value of “javax.xml.registry.security.authenticationMethod” is understood. The rest of the Connection properties defined in the JAXR specification are optional, and their values are not specific to the UDDI registry. The JAXR Provider for UDDI does not define any additional provider-specific properties.

Property	Description
javax.xml.registry.queryManagerURL	The URL of the UDDI registry’s inquiry API for UDDI Version 2. Typically this will be of the form: “http://<hostname>:<port>/uddisoap/inquiryapi”. This property is required.
javax.xml.registry.lifeCycleManagerURL	The URL of the UDDI registry’s publish API for UDDI v2. Typically this will be of the form: “http://<hostname>:<port>/uddisoap/publishapi”. If this property is not specified, it defaults to the value of the javax.xml.registry.queryManagerURL property, however the UDDI registry will typically have different URLs for the inquiry and publish APIs, and it is recommended to specifiy both properties.
javax.xml.registry.authenticationMethod	The method of authentication to use when authenticating with the registry. This may take one of two values, “UDDI_GET_AUTHTOKEN” and “HTTP_BASIC”. The default value is “UDDI_GET_AUTHTOKEN” if none is specified. See section Authentication and Security below for more information.

Authentication and security

Authentication

The javax.xml.registry.authenticationMethod Connection property tells the JAXR Provider which method to use when authenticating with the UDDI registry. The two supported values of this property are “UDDI_GET_AUTHTOKEN” and “HTTP_BASIC”. The JAXR Provider for UDDI does not support the “CLIENT_CERTIFICATE” or “MS_PASSPORT” methods of authentication. If this property is not set, the default authentication method is “UDDI_GET_AUTHTOKEN”.

UDDI_GET_AUTHTOKEN

The JAXR Provider uses the UDDI V2 get_authToken API to authenticate with the registry. The get_authToken call is made automatically by the JAXR Provider when the Connection credentials are set, and the UDDI V2 authToken returned by the call is saved by the JAXR Provider for use on subsequent UDDI publish API calls.

HTTP_BASIC

The JAXR Provider uses HTTP basic authentication to authenticate with the registry. This is supported by WebSphere Application Server when security is on. No UDDI V2 get_authToken API call is made, instead the username and password are sent in the HTTP headers using HTTP basic authentication every time a UDDI API call is made (both inquiry and publish). If the UDDI registry does not require HTTP basic authentication, the credentials are ignored.

The JAXR Provider uses UDDI Version 2 SOAP inquiry and publish APIs. These APIs are protected as described in Access control for UDDI registry interfaces.

USING SSL (Secure Sockets Layer)

SSL can be used to encrypt HTTP traffic between the JAXR Provider for UDDI and the UDDI registry. To use SSL, the JAXR client program should do the following:

1. When setting the “javax.xml.registry.queryManagerURL” and “javax.xml.registry.lifeCycleManagerURL” Connection properties, specify a URL with the protocol “https” and the correct port for using SSL to access the UDDI registry. The UDDI registry’s default port for HTTPS is 9443. Often only the lifeCycleManager URL (the UDDI Publish API URL) will require SSL.
2. Add a new Security Provider to the java.security.Security object, according to the JSSE (Java Secure Sockets Extension) implementation being used. If running under the JVM provided in WebSphere Application Server, the JSSE provided by IBM will automatically be on the classpath. Add the IBM Security Provider as follows:

   ```
   java.security.Security.addProvider(new com.ibm.jsse.JSSEProvider());
   ```
3. Set the System property “javax.net.ssl.trustStore” to be the file name of the client trust store file. The client trust store file is a Java key store (.jks) file and must contain the server certificate of the UDDI registry. Key store files can be managed using the ikeyman tool
4. Set the System property “javax.net.ssl.trustStorePassword”. This is the password used to open the client trust store file.
5. If using an IBM version of JVM that is older than the level provided in this version of WebSphere Application Server, it may be necessary to set the System property “java.protocol.handler.pkgs” to “com.ibm.net.ssl.internal.www.protocol”. For more information on SSL and the ikeyman tool refer to SSL and IKEYMAN within this Information Center.

Internal taxonomies

The JAXR Provider for UDDI supplies the following internal taxonomies:

Taxonomy	ClassificationScheme name (UDDI tModel name)	ClassificationScheme id (UDDI Version 2 tModelKey)
NAICS 1997	ntis-gov:naics:1997	UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2

NAICS 2002	ntis-gov:naics:2002	UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2
UNSPSC 3.1	unspsc-org:unspsc:3-1	UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384
UNSPSC 7	unspsc-org:unspsc	UUID:CD153257-086A-4237-B336-6BDCBDCC6634
ISO3166 2003	ubr-uddi-org:iso-ch:3166-2003	UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88

The tModels corresponding to all of these taxonomies are available in the UDDI Version 3 registry. If using the JAXR Provider to access a UDDI Version 2 registry, only the tModels corresponding to NAICS 1997, UNSPSC 3.1 and ISO3166 are available.

Custom internal taxonomies

A user may supply their own custom internal taxonomies. To create a new custom internal taxonomy and make it available to the JAXR provider, follow these steps:

1. Create a text file containing the taxonomy element data. As an example, look at the file iso3166-2003-data.txt in plugins/com.ibm.uddi_1.0.0. This is the taxonomy data file for the supplied ISO 3166 taxonomy. The first few lines are:

   ```
   iso3166#--#World#--
   iso3166#AD#Andorra#--
   iso3166#AE#United Arab Emirates#--
   iso3166#AE-AJ#'Ajm?n#AE
   iso3166#AE-AZ#Ab? Z?aby[Abu Dhabi]#AE
   iso3166#AE-DU#Dubayy [Dubai]#AE
   iso3166#AE-FU#Al Fujayrah#AE
   iso3166#AE-RK#Ra's al Khaymah#AE
   iso3166#AE-SH#Ash Sh?riqah [Sharjah]#AE
   iso3166#AE-UQ#Umm al Qaywayn#AE
   iso3166#AF#Afghanistan#--
   iso3166#AF-BAL#Balkh#AF
   iso3166#AF-BAM#B?m??n#AF
   ```

   Each line represents one element of the taxonomy, or one Concept in the taxonomy Concept tree. Each line has the form:

   ```
   <taxonomy ID>#<element value>#<element name>#<parent element value>
   ```

Token	Description
<taxonomy ID>	The taxonomy ID is the same for every element of a taxonomy.
<element value>	The Concept value (UDDI keyValue).
<element name>	The Concept name (UDDI keyName).
<parent element value>	This defines the element's parent element in the taxonomy tree. For every element (except the root element) in the data file, there should be another line which defines the element's parent element. The root element is denoted by defining itself as its own parent. There should be only one root element, and no parentless elements.
#	The delimiter character. This does not have to be "#" and can be defined for each taxonomy in the taxonomyConfig.properties file.

2. Save a ClassificationScheme (UDDI tModel) in the UDDI registry to represent the new internal taxonomy. This can be done using the javax.xml.registry.BusinessLifeCycleManager.saveClassificationSchemes() method.
3. Add the new taxonomy to the taxonomyConfig.properties file:
   a. Copy the supplied taxonomyConfig.properties file from the root of the com.ibm.uddi_1.0.0.jar file.

      The content of the supplied taxonomyConfig.properties file is:

```
naics-1997 = UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2, naics-1997-data.txt, #
naics-2002 = UUID:1FF729F2-1948-46CF-B660-31EC107F1663, naics-2002-data.txt, #
unspsc  = UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384, unspsc-data.txt,     #
unspsc7_data = UUID:CD153257-086A-4237-B336-6BDCBDCC6634, unspsc7-data.txt,    #
iso3166-2003 = UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88, iso3166-2003-data.txt,#
```

This file has one line per supplied internal taxonomy, which is of the form:

```
<taxonomy ID> = <tModelKey>,<data filename>,<data file delimiter>
```

Token	Description
<taxonomy ID>	This is used internally by the JAXR provider to identify each taxonomy. It does not have to be the same as the taxonomy ID in the corresponding taxonomy data file.
<tModelKey>	The tModelKey of the corresponding UDDI tModel. (The id of the corresponding JAXR ClassificationScheme).
<data filename>	The name of the corresponding taxonomy data file.
<data file delimiter>	The delimiter character used in the taxonomy data file. All supplied internal taxonomies use “#”, but user-supplied internal taxonomies may use different delimiters.

b. Add a new line for the new taxonomy to the copy of the taxonomyConfig.properties file. Do not remove any existing taxonomies from the file as this will make them unavailable to the JAXR provider.

4. Add the copied taxonomyConfig.properties file to the Java classpath ahead of jaxruddi.jar.
5. If any JAXR client programs are still running that were started before the new taxonomy was added to the taxonomyConfig.properties file, a new Connection must be created in order to pick up the new taxonomy.

Important notes on internal taxonomies

Each internal taxonomy is loaded into memory once per JAXR Connection. The taxonomy’s ClassificationScheme is created when the Connection is created. At this time the associated UDDI tModel is obtained from the registry and used to populate the ClassificationScheme attributes. The taxonomy’s Concept object tree is not created until the first time the ClassificationScheme is requested by the user. All subsequent requests for the same internal taxonomy using the same Connection will return the same object tree.

Modification of the Concept object tree

Because there is only one ClassificationScheme and Concept object tree per internal taxonomy per Connection, a user should not attempt to modify programmatically any part of the Concept tree, because all future requests for this taxonomy using the same Connection will return the modified (and now possibly invalid) objects. Programmatic modification of the Concept tree will not result in any changes to the associated taxonomy data file. If a user wishes to make a change to the values in a user-defined internal taxonomy, they must first make the changes in the taxonomy data file, and then create a new Connection to pick up the changes in a new Concept tree.

Modification of the ClassificationScheme

Similarly, a user should not attempt to modify programmatically an internal ClassificationScheme, except in the case where a user wishes to modify and then save a user-defined internal ClassificationScheme. A new Connection is not required to pick up programmatic changes.

Logging and messages

UDDI4J Logging

The JAXR Provider for UDDI uses UDDI4J Version 2 to communicate with the UDDI registry. UDDI4J has its own logging which can be switched on by setting the value of the System property "org.uddi4j.logEnabled" to "true". This outputs to the standard error log the XML request and response bodies of every UDDI request.

Trace

Entry, exit, exception, warning and debug trace is provided using commons-logging. See http://jakarta.apache.org/commons/logging/ for more information on commons-logging. Trace will only be created if the JAXR client configures it. Entry, exit and debug trace uses the debug level of logging. Exception and warning trace uses the info level of logging. Additionally, info level logging is provided before each UDDI4J request is made.

Standard error log messages

The InternalTaxonomyManager, EnumerationManager and PostalSchemeManager send warning messages to System.err if error conditions occur that do not warrant an exception, but that the user should be informed of. Examples of these are if a taxonomy data file contains an invalid line, or if a tModel corresponding to an internal taxonomy could not be found in the registry.

Removing and reinstalling the UDDI registry

A UDDI registry node consists of a J2EE application, a store of data (using a relational database management system) referred to as the UDDI database, and a means to connect the application to the data (a data source and related elements). All the data relating to UDDI is stored within the UDDI database and therefore exists irrespective of the UDDI application.

With these facts in mind, consider the following options:

- To remove a node from a WebSphere Application Server you do not have to delete the database. You only have to delete the UDDI application and any associated resources such as the data source used (and J2C Authentication Data if used), as the data in the UDDI database is separate from the UDDI application.
- You do not have to remove the UDDI application to start a new UDDI registry node. Instead you can create a new, replacement node by changing the datasource which the UDDI application uses to access the new UDDI database.

Remove the UDDI application if you no longer want a UDDI facility on a particular WebSphere Application Server (you can subsequently move the UDDI registry node to a different WebSphere Application Server).

Reinstall the UDDI application if you wish to continue to provide the UDDI facility on a particular WebSphere Application Server.

To reinstall a UDDI registry node, see Reinstalling the UDDI application. To remove a UDDI application or to remove a UDDI registry node, see Removing a UDDI Node.

Removing a UDDI registry node

To completely remove a UDDI registry node you need to remove the UDDI registry application and delete the UDDI registry database. However there may be situations where you only want to perform one of these tasks:

Removing the UDDI registry application from an application server.

You might want to do this in order to reinstall the application because it has become corrupted for some reason, or to apply service. See also the topic on Reinstalling the UDDI registry application.

Deleting the UDDI registry database

You might want to do this in order to use a different database product as the persistence store for your UDDI data, to delete all your UDDI registry data in order to publish fresh data (for example, if you have completed a test cycle), or to cause the UDDI registry node to re-initialize with new UDDI property settings (for example, to move from a default UDDI node to a customized UDDI node). Note that deleting a UDDI registry database will cause all UDDI data for that UDDI registry to be lost.

Moving a UDDI registry to another server or profile

You might want to do this if you have created a new profile and want to move the UDDI registry to it.

Completely removing a UDDI registry node from an application server

You might want to do this to remove a UDDI registry that has been used for testing.

Depending on what you wish to achieve, complete **one** of the following steps:

1. **Removing a UDDI registry application**

   To remove the UDDI application from an application server, run the wsadmin script uddiRemove.jacl, as shown, from the *app_server_root*/bin directory.

   The syntax of the command is as follows (this is a Windows platform example; for UNIX or Linux platforms, add the .sh suffix to the wsadmin command):

   ```
   wsadmin [-profileName profile_name] -f uddiRemove.jacl
                  node_name
                  server_name
                  [default]
   ```

   where
   - '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
   - *node_name* and *server_name* are the names of the WebSphere node and application server in which the UDDI application is deployed (these are the names that you specified when you ran uddiDeploy.jacl to install the UDDI application).
   - 'default' is optional and is applicable only for Cloudscape, and then only if the default option was used when the uddiDeploy.jacl script was run to deploy the UDDI registry. Specifying default will remove the UDDI Cloudscape datasource but **not** the UDDI Cloudscape database.

   Output will appear on the screen by default. To direct the output to a log file, add '*> removeuddi.log*' (where removeuddi.log can be any log name of your choice) to the end of the command.

   For example, to remove the UDDI application from server 'server1' running in node 'MyNode' on a Windows system, and send any messages to the file removeuddi.log:

   ```
   wsadmin -f uddiRemove.jacl MyNode server1 > removeuddi.log
   ```

   Note: You can also remove the UDDI registry application using the administrative console in the usual way, by selecting the application in the **Enterprise Applications** view and clicking **Uninstall**.

2. **Deleting a UDDI registry database**

   Note that this will cause all UDDI data in that UDDI registry to be destroyed.

   a. Stop the server that is hosting the UDDI registry application.

   b. Delete the database:
      - For DB2, use the database facilities to delete the UDDI database.
      - For Oracle, delete the schemas IBMUDDI, IBMUDI30 and IBMUDS30.
      - For Cloudscape, delete the directory tree containing the UDDI database. By default, this will be located in *app_server_root*/profiles/*profile_name*/databases/com.ibm.uddi/UDDI30.

3. **Moving a UDDI registry to another server or profile**

   a. Make sure that the UDDI database will still be accessible. For example if you are using a remote database make sure that the new server can access it. If your database will not be accessible, or it

will be deleted after the move, copy it to a new, accessible location. For example, if you want to move the UDDI registry to a new profile and then delete the old profile, any databases stored in the profile (such as a Cloudscape database created as part of the creation of a default UDDI node) will also be deleted.

b. Remove the UDDI registry application by running the uddiRemove.jacl script as described above.

c. If you ran the uddiRemove.jacl script using the default option, the datasource and related objects have already been deleted. If the default option was not valid for your configuration, delete the following objects:
   - the UDDI datasource that references the UDDI registry database (this was created when you set up the UDDI registry).
   - any UDDI JDBC provider that was created (if you did not reuse an existing JDBC provider).
   - any J2C Authentication Data Entry that was created.

d. In the new server create a J2C authentication data entry (if appropriate), JDBC provider and datasource, to reference the existing database (for more information see the relevant steps in Setting up a customized UDDI node).

e. Deploy the UDDI registry application as described in Deploying the UDDI registry application, noting that you should not specify the default option even if you previously used this option to set up a default UDDI node. If you use the default option, you might encounter an error during deployment, or in some circumstances your existing UDDI data might be overwritten.

   Note: The UDDI node name will remain the same as before. If the UDDI node name included the node name and server name of the original server, there will be a mismatch between the UDDI node name and the node name and server name of the new server. However this name mismatch will not affect the functioning of the UDDI registry node.

f. Check that the UDDI data can be accessed. If you are using a copy of the original UDDI registry database, you can now delete the original as described above.

4. **Completely removing a UDDI registry node**

   To completely remove a UDDI registry node from an application server, you need to remove the UDDI registry application and database, and also the resources that were used to reference the UDDI registry database.

   a. Remove the UDDI registry application by running the uddiRemove.jacl script as described above.

   b. Delete the UDDI registry database, as described above.

   c. If you ran the uddiRemove.jacl script using the default option, the datasource and related objects have already been deleted so no further action is required. If the default option was not valid for your configuration, delete the following objects:
      - the UDDI datasource that references the UDDI registry database (this was created when you set up the UDDI registry).
      - any UDDI JDBC provider that was created (if you did not reuse an existing JDBC provider).
      - any J2C Authentication Data Entry that was created.

Reinstalling the UDDI registry application

Perform this task if you want to continue providing UDDI services with your existing UDDI database, but require that the UDDI application code be changed.

1. Make a note of any changes that you have made to the installed UDDI application that you wish to keep, for example changes to security role mappings, changes to the deployment descriptor (web.xml) in v3soap,war, v3gui.war, v3soap.war, or soap.war, or customization of the UDDI user interface (GUI). All such changes will be lost during the reinstallation process; any changes that you wish to keep will need to be reapplied later.
2. Run the wsadmin script uddiDeploy.jacl from the *app_server_root*/bin directory, as shown, noting that:

- you should not specify the default option even if you previously used this option to set up a default UDDI node using Cloudscape. If you use the default option, you might encounter an error during deployment, or in some circumstances your existing UDDI data might be overwritten.
- if you are deploying the UDDI registry into a network deployment scenario, ensure that the deployment manager is the target.

At a command prompt enter:

```
wsadmin [-conntype none] [-profileName profile_name] –f uddiDeploy.jacl
                node_name
                server_name
```

where
- '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
- '-conntype none' is optional, and is only needed if the application server or deployment manger is not running.
- *node_name* and *server_name* are the names of the WebSphere node and application server in which the UDDI application is deployed (these are the names that you specified when you ran uddiDeploy.jacl to install the UDDI application).

The output from this command can optionally be redirected to a log file by adding '> *log_name*.log' at the end of the command, where *log_name*.log is the name of the log file to be created.

The command removes the existing UDDI application and reinstalls it.

Note: Your existing JDBC provider, datasource and any J2C authdata entry will be unchanged by this procedure . Your existing UDDI registry data, including UDDI entities as well as property and policy settings, will also be unaffected.

3. If you noted any changes in step 1, reapply them now.
4. Start or restart the application server for the reapplied changes to take effect.

Applying an upgrade to the UDDI registry

Perform this task to apply an iFix, Fix Pack or Refresh Pack to the UDDI registry.

Note: Some upgrades may require additional steps; refer to the readme file for the upgrade before you complete this task.

1. Apply the WebSphere Application Server iFix, Fix Pack or Refresh Pack to your application server or servers using the WebSphere Application Server Update Installer. Note that this process must be repeated for each server into which you choose to incorporate the UDDI upgrade.
2. If you have not yet deployed a UDDI registry, no further action is required as updates to the UDDI registry will take effect when you first deploy UDDI into any of your application server profiles.
3. If you have already deployed a UDDI registry to one or more application server profiles, redeploy the UDDI application as described in Reinstalling the UDDI registry application, to apply the upgrade. The existing UDDI application will be removed and the updated application will be deployed.

Configuring the UDDI registry application

The UDDI registry is supplied as a Java 2 Platform, Enterprise Edition (J2EE) application file, uddi.ear.

You can configure the following aspects of the UDDI registry:
- Configuring UDDI registry security
- Configuring SOAP API and GUI services
- Multiple language encoding support in UDDI
- Customizing the UDDI registry user interface (GUI)

Configuring UDDI registry security

The UDDI Version 3 registry is designed to exploit the advantages of WebSphere Application Server security. The registry also supports the UDDI Version 1 and Version 2 security features and the UDDI Version 3 Security API.

For production use, it is recommended that the UDDI Version 3 registry is configured to use WebSphere Application Server security, and that use of UDDI Version 1 and Version 2 security features and the Version 3 Security API is avoided. However, for solutions with a strong preference for UDDI security, the UDDI Version 3 registry can be configured to enable this, both when WebSphere Application Server security is enabled and when it is disabled.

To configure UDDI registry security, complete the following steps:

1. Follow the appropriate link for the type of configuration you wish to set up:
   - "Configuring the UDDI registry to use WebSphere Application Server security"
   - "Configuring the UDDI registry to use UDDI security" on page 508
2. Review the "UDDI registry security additional considerations" on page 510.

Configuring the UDDI registry to use WebSphere Application Server security

Before starting this task complete the following two steps:

- Enable WebSphere Application Server security (see Enabling security for all application servers). This will allow the UDDI registry to exploit the WebSphere Application Server security features.
- Ensure that WebSphere Application Server is configured to use HTTPS (SSL); this will allow the use of secure access with the UDDI registry. WebSphere Application Server is configured by default to accept SSL requests on port 9443, however if you need to make any additional SSL configuration changes, refer to SSL configurations for selected scopes.

There are two aspects of WebSphere Application Server security which are exploited by the UDDI registry:

Authorization
: Authorization determines whether users are allowed access to services. WebSphere Application Server determines authorization by mapping users, or groups of users, to roles. UDDI makes use of two WebSphere Application Server special subjects: *Everyone* (all users are allowed access) and *AllAuthenticatedUsers* (only valid WebSphere Application Server registered users are allowed access).

Data confidentiality
: Data confidentiality determines security at the transport level. Data confidentiality for WebSphere Application Server services can be either 'none' (HTTP is used as the transport protocol) or 'confidential' (requiring the use of SSL; HTTPS is used as the transport protocol).

When WebSphere Application Server security is enabled, the default settings in the UDDI Version 3 Application and Web deployment descriptors result in the following features:

- Publish, Custody Transfer and Security services are mapped to the AllAuthenticatedUsers special subject, and data confidentiality is enforced (HTTPS is used). Authentication uses the standard WebSphere Application Server security facilities and there is no separate registration function for the UDDI registry. To use publish functions, users must supply their WebSphere Application Server user name and password (unless you have modified the supplied publish role), and must also be registered UDDI Publishers. By registering users as UDDI Publishers you control which users in the AllAuthenticatedUsers subject can update the UDDI registry.
- Inquiry services are mapped to the Everyone special subject, and data confidentiality is not enforced (HTTP is used). To use inquiry services, users do not need to supply a user name or password, and do not have to be registered UDDI publishers.

You are recommended to use the default settings as described above. However, you can change the defaults by mapping roles to different users or user groups, or by not mapping a role to any users or user groups, in which case all access to that role will be disabled. If you do map roles to users or groups, turn on the **Automatically register UDDI publishers** property (see UDDI node settings) so that you do not have to use two mechanisms for giving access to a subset of users.

For more information about UDDI role mappings, and a list of UDDI registry services and roles, see Access control for UDDI registry interfaces.

To change the default settings follow the steps below:

1. To change the role mappings using the administrative console, complete the following steps:
   a. In the navigation pane, click **Applications** → **Enterprise Applications**.
   b. In the content pane, click the UDDI registry application.
   c. Under **Detail Properties** click **Security role to user/group mapping**.
   d. Make any changes you require and click **OK**.
2. To change the role mappings using the wsadmin command, complete the following steps:
   a. Use the MapRolesToUsers option of the edit command of the AdminApp object to map the roles defined in the UDDI registry application to special subjects (Everyone or AllAuthenticatedUsers), to users, or to user groups. For example, the following Java command language (JACL) statement maps the Version 3 GUI Publish role to Everyone, and the Version 3 SOAP Publish role to user 'user1' and group 'group1':

      ```
      $AdminApp edit $AppName {-MapRolesToUsers { {"GUI_Publish_User" Yes No "" ""} {"V3SOAP_Publish_
      User_Role" No No "user1" "group1"} }}
      ```

      where $AppName is a variable representing the name of the UDDI registry application.

      For more information about using the MapRolesToUsers option, see Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands.
3. To change the data confidentiality settings, refer to "Configuring SOAP API and GUI services" on page 512.

Configuring the UDDI registry to use UDDI security

It is possible to exploit the UDDI registry security features when WebSphere Application Server security is either enabled or disabled. Each situation requires different configuration, and different behavior is achieved.

Note: While useful for test purposes, it is not anticipated that WebSphere Application Server security is disabled for production configurations.

To continue configuring the UDDI registry to use UDDI security, choose one of the following options:

- "Configuring UDDI Security with WebSphere Application Server security enabled"
- "Configuring UDDI Security with WebSphere Application Server security disabled" on page 509

Configuring UDDI Security with WebSphere Application Server security enabled:

When WebSphere Application Server security is enabled, to use the UDDI Version 1 and Version 2 publish security features (use of authentication tokens) or the UDDI Version 3 security API, use the administrative console to complete the following steps:

1. In the navigation pane, click **Applications** → **Enterprise Applications**.
2. In the content pane, click the UDDI registry application. Under **Detail Properties** click **Security role to user/group mapping**.
3. Set the WebSphere Application Server security role mappings to Everyone for the following UDDI services:

- Versions 1 and 2 SOAP publish service (SOAP_Publish _User)
- Version 3 publish service (V3SOAP_Publish_User_Role)
- Version 3 custody transfer service (V3SOAP_CustodyTransfer_User_Role)
- Version 3 security service (V3SOAP_Security_User_Role)

Changing the role mappings to Everyone prevents WebSphere Application Server security from overriding UDDI security.

4. Ensure that UDDI Policy is set to require the use of authentication tokens for the UDDI Version 3 Publish and Custody Transfer services (use of authentication tokens is already required for Version 1 and Version 2 Publish services). To do this, click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups** click **API policies**. Select the **Authorization for publish** and **Authorization for custody transfer** check boxes. (Select the **Authorization for inquiry** check box if you require authentication for UDDI Inquiry services).
5. Click **OK**.

With this configuration, no Security Role authentication restriction is imposed, but the credentials (user name and password) associated with the authentication token are authenticated by WebSphere Application Server.

Note: When WebSphere Application Server security is enabled, WebSphere Application Server data confidentiality management is independent of UDDI security and is managed as described in "Configuring the UDDI registry to use WebSphere Application Server security" on page 507.

Configuring UDDI Security with WebSphere Application Server security disabled:

With WebSphere Application Server security disabled, neither WebSphere Application Server security roles nor data confidentiality constraints apply. This mode may be useful for test UDDI registry configurations.

In this mode, UDDI Version 1 and Version 2 security features are active:

- UDDI Version 1 and Version 2 publish requests require UDDI Version 1 and Version 2 authentication tokens respectively. Publishers requesting or using an authentication token must be registered WebSphere Application Server users.
- UDDI Version 1 and Version 2 inquiry requests do not require authentication tokens.

No further configuration is required for UDDI Version 1 and Version 2 security.

For UDDI Version 3, the use of the UDDI Version 3 Security API, and the use of authentication tokens with Version 3 Publish and Custody Transfer APIs, is optional. To make use of these UDDI Version 3 security features, use the administrative console to complete the following steps:

1. Specify that use of authInfo is required. Click **UDDI** → **UDDI Nodes** > *uddi_node_name*.
2. In the **General Properties** section, select the **Use authInfo credentials if provided** check box.
3. Click **OK**.

Authentication tokens will now be required for publish and custody transfer requests, but not for inquiry requests. Publishers requesting or using an authentication token must be registered WebSphere Application Server users.

Access control for UDDI registry interfaces

Access to UDDI registry interfaces is controlled by a combination of J2EE declarative security using role mappings, and UDDI properties and policies such as the registering of users as UDDI publishers.

Each of the UDDI registry interfaces is represented by a security role. The interfaces and their corresponding roles are as follows:

UDDI registry interface	Security role
Version 3 SOAP inquiry	V3SOAP_Inquiry_User_Role
Version 3 SOAP publish	V3SOAP_Publish_User_Role
Version 3 SOAP custody transfer	V3SOAP_CustodyTransfer_User_Role
Version 3 SOAP security	V3SOAP_Security_User_Role
Version 3 GUI inquiry	GUI_Inquiry_User
Version 3 GUI publish	GUI_Publish_User
Versions 1 and 2 SOAP inquiry	SOAP_Inquiry_User
Versions 1 and 2 SOAP publish	SOAP_Publish_User
EJB inquiry	EJB_Inquiry_Role
EJB publish	EJB_Publish_Role

By default, the inquiry roles are mapped to the *Everyone* special subject and the non inquiry roles are mapped to the *AllAuthenticatedUsers* special subject. For more information about WebSphere Application Server role mapping and the Everyone and AllAuthenticatedUsers special subjects, see Role-based authorization. With these default settings, after you enable WebSphere Application Server security you do not need access control to use the UDDI registry inquiry interfaces, however to use the publish roles and the Version 3 custody transfer role you must be authenticated using a WebSphere Application Server userid and password. (The Version 3 security role is a special case, as this concerns use of UDDI registry security instead of WebSphere Application Server security, and must be specially configured as described in Configuring the UDDI registry to use UDDI security.)

For more information about UDDI registry security roles and how they can be used to control authorization and data confidentiality, see Configuring the UDDI registry to use WebSphere Application Server security.

Roles which are mapped to AllAuthenticatedUsers (as the UDDI registry publish interfaces are by default) are further protected in that, having successfully authenticated, the user must also be registered as a UDDI publisher in order to publish data to the UDDI registry. An E_unknownUser error is returned in the disposition report if the user is not registered. You can register users as UDDI publishers in one of two ways:

- Create a new UDDI publisher using the administrative console (see UDDI Publisher collection) or the JMX interface (see UDDI registry Administrative (JMX) Interface).
- Set the **Automatically register UDDI publishers** property (see UDDI node settings), in which case users will be automatically registered as a publisher on their first publish request.

In accordance with the UDDI specification, there is additional access control in that an entity which has been published to the UDDI registry can only be updated or deleted by the user who originally published that entity.

The UDDI registry also provides some management interfaces which are protected by requiring administrative permissions for certain operations; see UDDI registry Management Interfaces for details.

UDDI registry security additional considerations

In addition to the configuration of UDDI registry security, there a number of other UDDI registry settings which may affect the behavior of the UDDI registry. Some of these settings are security specific, others are points to bear in mind when configuring security.

Additional security considerations

UDDI registry interfaces are protected as detailed in Access control for UDDI registry interfaces.

The UDDI registry supports the use of XML Digital Signatures to sign UDDI entities. This is described in Use of digital signatures with the UDDI registry.

Additional policy considerations

A number of the UDDI property and policy settings also determine the behavior of a UDDI registry with respect to security.

To review or change the following property settings, click **UDDI** → **UDDI Nodes** > *uddi_node_name*. The settings are also detailed in the administrative console help.

Key space requests require digital signature
This setting determines whether all tModel:keyGenerator requests for key space must be digitally signed. To understand key space refer to UDDI registry Version 3 Entity Keys.

Use authInfo credentials if provided
This setting applies only when WebSphere Application Server security is disabled. See Configuring UDDI Security with WebSphere Application Server security disabled.

Authentication token expiry period
The authentication token expiry period is the length of idle time (in minutes) allowed before an authentication token becomes invalid.

Default user name
The default user name is used for publish operations when WebSphere Application Server security is disabled and no authentication token data is supplied.

To review or change the following policy settings, click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups**, click **API policies**. The settings are also detailed in the administrative console help.

Authorization for inquiry
Specifies whether authorization using authentication tokens is required for inquiry API requests.

Authorization for publish
Specifies whether authorization using authentication tokens is required for publish API requests.

Authorization for custody transfer
Specifies whether authorization using authentication tokens is required for custody transfer API requests.

The above policy settings apply when UDDI security features are being used and WebSphere Application Server security is enabled. If the UDDI service in question is mapped to the security role AllAuthenticatedUsers, these settings will be overridden. See Configuring UDDI Security with WebSphere Application Server security enabled.

Other considerations

The publish related actions that a registered UDDI publisher can perform are defined by their entitlements, as described in UDDI registry user entitlements.

In addition to the property and policy settings above, be aware that some UDDI keying and user policy settings also influence publish behavior. These settings are not specific to security, but you should bear them in mind as they also place restrictions on successful completion of publish requests.

To review or change the following property settings, click **UDDI** → **UDDI Nodes** > *uddi_node_name*. The settings are also detailed in the administrative console help

Automatically register UDDI publishers
The UDDI registry requires publisher entitlements to be set before allowing any publish requests. This option automatically registers users with default entitlements.

If this option is not selected, users (and their entitlements) can be registered. See UDDI Publisher settings.

Use tier limits
If selected, tier limits are enforced.

If this option is selected you should have one or more tiers configured (see Tier collection and UDDI Tier settings). You should also ensure that registered UDDI Publishers are assigned to a tier (see UDDI Publisher settings).

To review or change the following property setting, click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups** click **Keying policies**. The setting is also detailed in the administrative console help.

Registry key generation
If this option is selected, publishers may request key space and, if successful, publish with publisher assigned keys.

UDDI registry user entitlements: UDDI registry *user entitlements* define the set of publish related actions that registered UDDI users are entitled to perform.

An important entitlement is the number of entities of each type that a UDDI user is entitled to publish; this is controlled by assigning the user to a UDDI publisher *tier*. Any number of tiers can be defined to the UDDI registry, and there are some predefined tiers which are supplied when the UDDI registry is deployed. A UDDI publisher tier specifies the maximum number of each entity (business, service, binding, tModel, and publisher assertion) that a user assigned to that tier may publish. (See UDDI Tier settings for information about defining UDDI publisher tiers.)

Other entitlements relate to the user's entitlement to allocate key spaces within which they can specify publisher assigned keys when publishing UDDI entities. A key space is allocated by publishing a keyGenerator tModel, and there are a number of entitlements relating to different kinds of key generator. For full details of these entitlements, see UDDI Publisher settings. For more information about key generators and publisher assigned keys, see UDDI registry Version 3 Entity Keys.

The entitlements for a UDDI user can be set using the administrative console, or through JMX using the UDDI Administrative interface, as described in UDDI node collection and UDDI registry Administrative (JMX) Interface.

Configuring SOAP API and GUI services

Configuring Version 1 and Version 2 SOAP API services

You can configure the following Version 1 and Version 2 SOAP interface properties:

- *defaultPoolSize*. This is the number of SOAP parsers with which to initialize the parser pool for the SOAP interface. You can set this independently for the Publish (uddipublish) and Inquiry (uddi) APIs. For example, if you expect more inquiries than publish requests through the SOAP interface, you can set a larger pool size for the Inquiry API. The default initial size for both APIs is 10.
- Whether the API is to be secure (accessed using HTTPS) or insecure (accessed using HTTP). The default for Publish is to use HTTPS and Inquiry to use HTTP.

To configure these properties after the UDDI application has been installed:

1. Edit the active deployment descriptor (web.xml) for the Version 1 and Version 2 SOAP module (soap.war).

   This file is located in the following directory:

   ```
   app_server_root/profiles/profile_name/config/cells/cell_name/applications/
     UDDIRegistry.node_name.server_name.ear/deployments/
       UDDIRegistry.node_name.server_name/soap.war/WEB-INF
   ```

2. To modify defaultPoolSize for Version 1 or Version 2 Publish, modify the 'param-value' element in the servlet with 'servlet-name' = uddipublish
3. To modify defaultPoolSize for Version 1 or Version 2 Inquiry, modify the 'param-value' element in the servlet with 'servlet-name' = uddi
4. To modify the user data constraint transport guarantee for Version 1 or Version 2 publish, which determines whether the Publish service is to be confidential (accessed using HTTPS) or insecure (using HTTP), find the 'security-constraint' with id = 'UDDIPublishTransportConstraint' and set its 'user-data-constraint' 'transport-guarantee' to CONFIDENTIAL or NONE.
5. Stop and restart the application server for the changes to take effect.

Configuring Version 3 SOAP API services

For the Version 3 SOAP interface, you can specify whether the Publish, Custody Transfer, Security and Inquiry API services are to be secure (accessed using HTTPS) or insecure (accessed using HTTP). The default for Publish, Custody Transfer and Security APIs is to use HTTPS, and Inquiry to use HTTP.

To configure these properties after the UDDI application has been installed:

1. Edit the active deployment descriptor (web.xml) for the Version 3 SOAP module (v3soap.war).

   This file is located in the following directory:

   app_server_root/profiles/*profile_name*/config/cells/*cell_name*/applications/
   UDDIRegistry.*node_name*.*server_name*.ear/deployments/
   UDDIRegistry.*node_name*.*server_name*/v3soap.war/WEB-INF
2. Each type of service is represented in the deployment descriptor (web.xml) file by a <security-constraint> element, which contains, amongst other things, a <display-name> element and a <user-data-constraint> <transport-guarantee> element. Locate the <security-constraint> element for the service that you want to modify by searching the deployment descriptor (web.xml) file for the relevant <display-name> value (see the table below). When you have found the <security-constraint> element, set its <user-data-constraint> <transport-guarantee> to either CONFIDENTIAL (the service can only be accessed using HTTPS) or NONE (the service can be accessed using HTTP).

Type of UDDI service	Value of <security-constraint> <display-name> element
Publish	AxisServlet Publish Resource Collection
Custody transfer	AxisServlet CustodyTransfer Resource Collection
Security	AxisServlet Security Resource Collection
Inquiry	AxisServlet Inquiry Resource Collection

3. Stop and restart the application server for the changes to take effect.

Configuring Version 3 GUI services

For the Version 3 GUI interface, you can specify whether the Publish and Inquiry API services are to be secure (accessed using HTTPS) or insecure (accessed using HTTP). The default for Publish is to use HTTPS, and Inquiry to use HTTP.

To configure these properties after the UDDI application has been installed:

1. Edit the active deployment descriptor (web.xml) for the Version 3 GUI module (v3gui.war).

   This file is located in the following directory:

   app_server_root/profiles/*profile_name*/config/cells/*cell_name*/applications/
   UDDIRegistry.*node_name*.*server_name*.ear/deployments/
   UDDIRegistry.*node_name*.*server_name*/v3gui.war/WEB-INF
2. Use the table below to find the id value for the <user-data-constraint> element that corresponds with the service that you want to change. Search for this value in the deployment descriptor (web.xml) file

to find the <user-data-constraint> element, then set the <user-data-constraint> <transport-guarantee> to either CONFIDENTIAL (the service can only be accessed using HTTPS) or NONE (the service can be accessed using HTTP).

Type of UDDI service	Value of <user-data-constraint id>
Publish	UDDIPublishTransportConstraint
Inquiry	UDDIInquireTransportConstraint

3. Stop and restart the application server for the changes to take effect.

Multiple language encoding support in UDDI

UDDI API

UDDI Version 3 supports both UTF-8 and UTF-16 encoding. Internally UTF-16 characters are stored as UTF-8. This is transparent to the user application.

UDDI User Console

The UDDI user console only supports UTF-8 encoding. To enable this, you must configure the application server into which the UDDI registry application is installed with UTF-8 encoding enabled. To do this, refer to ″Configuring application servers for UTF-8 encoding″ elsewhere in the WebSphere Information Center.

Customizing the UDDI registry user interface (GUI)

The look and feel of the UDDI registry user interface is determined by the styles defined in the *.css* files located in the following directory: *app_server_root*/profiles/*profile_name*/installedApps/*cell_name*/ UDDIRegistry.*node_name*.*server_name*.ear/v3gui.war/theme.

Style class definitions in these files can be edited to alter the overall theme of the UDDI registry user interface, including font attributes, layout and colors.

Managing the UDDI registry

You can use either the WebSphere Application Server administrative console or the Java Management Extensions (JMX) management interface to manage UDDI Registries.

In previous versions of WebSphere Application Server and the UDDI registry, a properties file was used, but from WebSphere Application Server Version 6, you manage all the policies and properties of the UDDI registry through either the JMX management interface or the administrative console.

JMX can be used to monitor and configure UDDI registries programmatically, and is explained in Using administrative programs (JMX). See UDDI registry Administrative (JMX) Interface for full details on using the UDDI administrative interface. To manage UDDI registries using the WebSphere Application Server administrative console, start from the UDDI link in the left navigation pane as described below.

Using the UDDI management functions available in the WebSphere Application Server administrative console, you can perform the following operations:

- view and manage the status of all UDDI nodes in a cell
- initialize UDDI nodes with required settings
- configure general properties that affect UDDI runtime behavior
- manage UDDI policy settings
- create, view and update UDDI publishers

- create, view and update publisher tiers which limit how many UDDI entries may be published
- view and manage the status of value sets

If WebSphere Application Server security is enabled, you will need to log in to the WebSphere Administrative Console, supplying a valid userid and password, in order to use the UDDI management functions.

UDDI node collection

Use this page to manage the UDDI nodes in this cell. Each UDDI node represents an individual UDDI registry application. A UDDI node will only appear in this list if its underlying UDDI application is started. The status of the UDDI node indicates whether the node is activated (available to accept API requests), deactivated (not allowing user requests), or not initialized. UDDI nodes that are not initialized require some properties to be set before they can be initialized and activated.

To view this administrative console page, click **UDDI** → **UDDI Nodes**.

Each UDDI node is represented by a UDDI Node ID, Description, UDDI Application Location and Status. To manage an individual UDDI node, click on its UDDI Node ID link. This takes you to the Configuration page where you can manage its general properties, initialize it if the status is set to *Initialization Pending*, and access pages for managing policies, UDDI publishers, tiers and value sets.

UDDI Node ID

Specifies the identifier for the UDDI node.

Description

Specifies the description of the UDDI node.

UDDI Application Location

Specifies the server in which the UDDI registry application is deployed and running.

For a cluster configuration, which will include several servers all running the UDDI registry application, you will see a single location listed; this location could represent any of the active servers.

Status

Specifies the status of the UDDI node.

The Status of the UDDI node can be *Not initialized*, *Initialization pending*, *Initialization in progress*, *Migration in progress*, *Migration pending*, *Value set creation in progress*, *Value set creation pending*, *Activated* or *Deactivated*. If a node is in the *Initialization pending* state, you must initialize it before you can activate it. If you attempt to initialize the node and it remains in a pending state, an error occurred during migration or initialization.

To activate UDDI nodes that are *Deactivated*, select them by checking the corresponding check boxes in the Select column and click **Activate**. Similarly to deactivate UDDI nodes, select them and click **Deactivate**.

Note: Restarting the UDDI application, or the application server, will always result in the reactivation of the UDDI node, even if the node was previously deactivated.

UDDI node settings

This topic contains details of the general properties that you can configure for a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id*.

By clicking on a node in the UDDI node ID column the UDDI node detail page is displayed. The UDDI node detail page displays a set of General Properties for the UDDI node, some of which may be editable

depending on the state of the node. There are also links to Additional Properties (Value sets, Tiers and UDDI Publishers) and links to Policy Groups where you can view and change UDDI node policy.

Unless you installed the UDDI node as a default UDDI node (as defined in UDDI registry terminology), there are some important general properties that you must set before you can initialize the UDDI node. These properties are marked as being required (indicated by the presence of a '*' next to the input field). You may set the values as many times as you wish before initialization. However, after you initialize the UDDI node, these properties will become read only for the lifetime of that UDDI node. It is very important to set these properties correctly. You can set other general properties of the UDDI node before, and after, initialization.

After you set the general properties to appropriate values, click **OK** to save your changes and exit the page, or **Apply** to save your changes and remain on the same page. At this point the changes are stored.

If the UDDI node is in the *Not Initialized* state, indicated on the UDDI node detail page by the presence of an **Initialize** button (above the General Properties section), you can initialize UDDI node by clicking **Initialize**. This operation can take a while to complete. It is important to save any changes you made to the general properties by clicking **Apply** or **OK**, before you click **Initialize**.

UDDI Node ID:

Specifies the unique identifier that is given to a UDDI node in a UDDI registry. The node ID must be a valid UDDI key. The value will also be the domain key for the UDDI node.

The value for the node ID will also be the domain key for this UDDI node.

Required	Yes
Data type	String
Default	uddi:*cell_name*:*node_name*:*server_name*:node_id

UDDI node description:

Specifies the description of this UDDI node.

Required	Yes
Data type	String
Default	WebSphere UDDI registry default node

Root key generator:

Specifies the root key space of the registry. Registries intending to become affiliate registries might want to specify a root key space in a partition below the root key generator of the parent root registry, for example, uddi:thisregistry.com:keygenerator.

Required	Yes
Data type	String
Default	uddi:*cell_name*:*node_name*:*server_name*:keyspace_id:keygenerator

Prefix for generated discoveryURLs:

Specifies the URL prefix that is applied to generated discoveryURLs in businessEntity elements, so they can be returned on HTTP GET requests. This property applies to UDDI version 2 API requests only. Set this prefix to a valid URL for your configuration, and do not change it unless absolutely necessary.

The format is `http://hostname:port/uddisoap/`, where `uddisoap` is the context root of the UDDI version 2 SOAP servlet.

Although this field is not required, you should set it so that the required and valid URL is generated in response to version 2 GET requests. After you have set the prefix you should not alter it unless a subsequent configuration change renders it invalid; if you change the prefix, discoveryURLs generated using the earlier prefix will no longer work.

Required	No
Data type	String
Default	http://localhost:9080/uddisoap

Host name for UDDI node services:

Specifies the host name root used by the UDDI node to model API services in its own node business entity. This value must be the fully qualified domain name, or IP address, of the network host.

The UDDI node provides Web services that implement each of the UDDI API sets that it supports. The host name is used to generate access point URLs in the bindingTemplate elements for each of the services. The access point URL is generated by prefixing the host name value with a protocol, such as http, and suffixing it with the corresponding host port number. The access point URL must resolve to a valid URL.

Data type	String
Default	localhost

Host HTTP port:

Specifies the port number used to access UDDI node services with HTTP. This port number must match the WebSphere Application Server port for HTTP requests.

Data type	Integer
Default	9080

Host HTTPS port:

Specifies the port number used to access UDDI node services with HTTPS. This port number must match the WebSphere Application Server port for HTTPS requests.

Data type	Integer
Default	9443

Maximum inquiry result set size:

Specifies the maximum size of result set which the registry will process for an inquiry API request.

If the result set exceeds this value, an E_resultSetTooLarge error is returned to the user. If you set this value too low, and users use imprecise search criteria, the likelihood of receiving an E_resultSetTooLarge error is increased. Setting the value higher allows larger result sets but can cause increased response times.

Data type	Integer
Default	500
Range	0 to 1024

Maximum inquiry response set size:

Specifies, for inquiry API requests, the maximum number of results returned in each response. Do not set this value higher than the value for **Maximum inquiry results**.

If the result set contains more results than this value, the response will include only a subset of those results. The user can retrieve the remaining results using the listDescription feature as described in the UDDI specification. If you set this value too low, the user has to make more requests to retrieve the remainder of the result set.

Data type	Integer
Default	500
Range	0 to 1024

Maximum search names:

Specifies the maximum number of names that can be supplied in an inquiry API request. To avoid slowing UDDI node response times, set this value no higher than 8.

Higher values allow more complex requests to be processed by the UDDI node, however complex requests can significantly slow the response times of the UDDI node.

Data type	Integer
Default	5
Range	1 to 64

Maximum search keys:

Specifies the maximum number of keys that can be supplied in an inquiry API request. To avoid slowing UDDI node response times, set this value no higher than 5.

This value limits the number of references that can be specified in categoryBag, identifierBag, tModelBag and discoveryURLs.

Higher values allow the UDDI node to process more complex requests, however complex requests can significantly slow the response times of the UDDI node. In exceptional cases, the UDDI node may reject complex requests with excessive numbers of keys even if the value of maxSearchKeys is not exceeded.

Data type	Integer
Default	5
Range	1 to 64

Key space requests require digital signature:

Specifies whether tModel:keyGenerator requests must be digitally signed.

Data type	Boolean (check box)
Default	False (cleared)

Use tier limits:

Specifies whether an approval manager is used to check publication tier limits. If you set this value to false, the number of UDDI entities that can be published is unlimited.

Data type	Boolean (check box)
Default	True (selected)

Use authInfo credentials if provided:

Specifies whether authInfo contents in UDDI API requests are used to validate users when WebSphere Application Server security is off. If you set this value to true, the UDDI node will use the authInfo element in the request, otherwise the default user name is used.

Data type	Boolean (check box)
Default	True (selected)

Authentication token expiry period:

Specifies the period, in minutes, after which an authentication token is invalidated and a new authentication token is required.

Set this value high enough to allow the registry to operate successfully, but be aware that high values can increase the risk of illegal use of authentication tokens.

Data type	Integer
Default	30
Range	1 to 10080 minutes (10080 minutes = 1 week)

Automatically register UDDI publishers:

Specifies whether UDDI publishers are automatically registered and assigned to the default tier. Automatically registered UDDI publishers are given default entitlements.

Data type	Boolean (check box)
Default	True (selected)

Default user name:

Specifies the user name used for publish operations when WebSphere Application Server security is disabled and **Use authInfo credentials if provided** is set to false.

Data type	String
Default	UNAUTHENTICATED

Default language code:

Specifies, for UDDI version 1 and version 2 requests, the default language code to be used for xml:lang, when not otherwise specified.

Data type	String
Default	en

Value set collection:

Use this page to view and configure the value sets that are installed in a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Value Sets**.

Value sets in a UDDI node are either supported or not supported by policy. By default, new value sets are *not* supported. After you publish a value set tModel and loaded value set data, you can control whether other UDDI entities can reference this value set tModel by setting the *Supported* policy.

To enable support for one or more value sets, select the value sets by selecting the appropriate check boxes in the **Select** column. Click **Enable Support**. The Supported field for all the selected value sets is updated, with a value of true, to reflect the new status.

To disable support for a value set, which might be necessary before you remove it from the UDDI node, select the value sets in the same manner as for enabling support. Click **Disable Support**. The Supported field for all the selected value sets is updated, with a value of false, to reflect the new status.

Clicking on a value set name in the list takes you to the general properties page for that value set as described in Value set settings.

Name:

Specifies the name of the tModel that represents this value set.

tModelkey:

Specifies the key for the tModel that represents this value set.

Supported:

Specifies whether this value set is supported by policy in this UDDI node.

Value set settings:

Use this page to view the attributes of a value set in a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Value Sets** > *value_set_name*.

This page shows the values of keyedReferences in the tModel that represents this value set. This page also shows the *Supported* status of the value set. All properties are read-only. To change the *Supported* status, use the Value sets collection page.

Unvalidatable:

Specifies whether this value set is unvalidatable. The value set tModel publisher sets this value, to indicate whether the value set is available for use by publish requests.

Checked:

Specifies whether this value set is checked. If you set this value to true, UDDI entities that reference this value set will be validated to ensure that their values are present in this value set.

Cached:

Specifies whether this value set is cached in this UDDI node.

Externally cacheable:

Specifies whether this value set is externally cacheable.

Externally validated:

Specifies whether this value set is externally validated.

Supported:

Specifies whether this value set is supported by policy in this UDDI node.

Last cached:

Specifies the date when this value set was last cached in the UDDI node.

Tier collection:

This page contains a list of the available tiers for the UDDI node. You can modify tiers, create new tiers and delete tiers from this page.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Tiers**.

This page shows the available tiers for the UDDI node. Clicking a tier name will show the General Properties for the specific tier as detailed in Tier settings. To delete a Tier from the list, select the relevant name and click *Delete*. Clicking *New* will take you to the General Properties page with the same properties as described in Tier settings.

One of the tiers in the collection will be marked as the default tier, indicated by *(default)* appearing next to the tier's name. The default tier will be assigned to UDDI publishers that are registered automatically when automatic user registration is turned on. To set the default tier, select the appropriate tier in the collection and press the *Set default* button. Note that it is not possible to delete a tier if it is currently marked as the default tier, or it is currently assigned to a UDDI publisher.

Name:

Specifies the name of the tier.

Description:

Specifies the descriptive text about the tier.

UDDI Tier settings:

This topic contains details of the general properties that you can configure for a UDDI publisher tier.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Tiers** > *tier_name*.

Name:

Specifies the name of the tier.

Required	Yes
Data type	String
Default	No default
Range	1 to 255

Description:

Specifies a description of the purpose or usage of this tier.

Data type	String
Default	No default
Range	0 to 255

Maximum properties:

For each of the maximum fields described below, the data is:

Required	Yes
Data type	Integer
Default	No default
Range	0 to 2147483647

Maximum businesses:

Specifies the maximum number of businesses that UDDI publishers in this tier are allowed to publish.

Maximum services per business:

Specifies the maximum number of services that UDDI publishers in this tier are allowed to publish for each business.

Maximum bindings per service:

Specifies the maximum number of bindings that UDDI publishers in this tier are allowed to publish for each service.

Maximum tModels:

Specifies the maximum number of tModels that UDDI publishers in this tier are allowed to publish.

Maximum publisher assertions:

Specifies the maximum number of publisher assertions that UDDI publishers in this tier are allowed to add.

UDDI Publisher collection:

This page shows the users that are currently registered as UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes >** *UDDI_node_id* **> [Additional Properties] UDDI Publishers**.

To create a UDDI publisher, click **New**. The “UDDI Publisher settings” on page 524 page, where you can enter details about the publisher, is displayed.

To register one or more existing WebSphere Application Server users as UDDI publishers, click **Create publishers**. The “Create UDDI Publishers” on page 523 page, where you can select users and modify their entitlements, is displayed.

You can assign multiple publishers to a tier without editing each publisher individually, by completing the following steps:

1. Select the appropriate publishers in the collection table.

2. From the tier list at the top of the collection table, select one of the tiers that is available on the UDDI node.
3. Click **Assign tier** to update the selected publishers.

To delete publishers, select them in the collection table and then click **Delete**.

After you register the users as UDDI publishers, you can edit their entitlements as described in “UDDI Publisher settings” on page 524.

User name:

Specifies the name of the UDDI publisher.

Tier:

Specifies the tier to which the UDDI publisher is assigned.

Create UDDI Publishers:

Use this page to register one or more existing WebSphere Application Server users as UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes >** *UDDI_node_id* **> [Additional Properties] UDDI Publishers** → **Create publishers**.

To register as UDDI publishers one or more users that are known to the application server, complete the following steps:

1. Enter a string to search for the required users. Use ’*’ to find all users.
2. Optionally, enter a number in the limit field to restrict the number of returned results.
3. Click **Search** to display a list of users that match the string.
4. Select the users that you want to register from the **Available** list and use the arrows to move them into the **Selected** list.
5. Use the entitlements listed under **General Properties** to give the UDDI publishers permission to perform specific actions. Set the entitlements by selecting the check box next to each entitlement.
6. Select a tier for the users from the **Tier** list.
7. Click **OK** to register the users as UDDI publishers with the specified entitlements and tier.

Note: If you are using a Lightweight Directory Access Protocol (LDAP) user registry, the format of the name that is given to each UDDI Publisher is defined by the **User ID map** value in the LDAP advanced settings. To view the LDAP advanced settings, click **Security** → **Secure administration, applications, and infrastructure**, under **User account repository** select **Standalone LDAP registry** and click **Configure**, then under **Additional Properties** click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

After a user has been registered as a UDDI publisher, you can edit their entitlements by clicking the user name.

Allowed to publish keyGenerator with derived key:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a derived key.

The tModel:keyGenerator is a request for key space. An example of a legal derived key is uddi:tempuri.com:fish:buyingService where the key is based on the derivedKey ″uddi:tempuri.com:fish″. the string ’buyingService’ is the key’s key specific string (KSS).

Allowed to publish keyGenerator with domain keys:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a domain key.

Allowed to publish keyGenerator:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator.

If false, UDDI publishers cannot publish keyGenerators of any kind. In this situation all the entitlement settings are disregarded, regardless of how they are set.

Allowed to publish with UUID key:

Specifies whether the UDDI publisher has permission to publish elements with a UUID key.

Allowed to publish keyGenerator with UUID keys:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a UUID key.

Tier:

Specifies the tier to which the UDDI publisher is assigned.

UDDI Publisher settings:

Use this page to view and edit the properties of a UDDI publisher, or to create a new UDDI publisher.

You can view this administrative console page in two ways:

- If you want to view and edit the properties of an existing UDDI publisher click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **UDDI Publishers** > *user_name*
- If you want to create a new UDDI publisher click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **UDDI Publishers** → **New**

This page shows the entitlements and publication limits tier for a particular UDDI publisher.

User name:

Specifies the name of the UDDI publisher.

If you are creating a new UDDI publisher, enter the name of a user known to the application server. If you are viewing or editing the properties of an existing publisher, the user name cannot be changed.

Allowed to publish keyGenerator with derived key:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a derived key.

The tModel:keyGenerator is a request for key space. An example of a legal derived key is uddi:tempuri.com:fish:buyingService where the key is based on the derivedKey ″uddi:tempuri.com:fish″. the string 'buyingService' is the key's key specific string (KSS).

Data type	Boolean
Default	True (selected)

Allowed to publish keyGenerator with domain keys:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a domain key.

Data type Boolean
Default True (selected)

Allowed to publish keyGenerator:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator.

If you set this value to false, the UDDI publisher cannot publish keyGenerators of any kind. In this situation the following permissions will be disregarded irrespective of how they are set: **Allowed to publish keyGenerator with derived key**, **Allowed to publish keyGenerator with domain keys**, **Allowed to publish with UUID key** and **Allowed to publish keyGenerator with UUID keys**.

Data type Boolean
Default True (selected)

Allowed to publish with UUID key:

Specifies whether the UDDI publisher has permission to publish elements with a UUID key.

Data type Boolean
Default False (cleared)

Allowed to publish keyGenerator with UUID keys:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a UUID key.

Data type Boolean
Default False (cleared)

Tier:

Specifies the tier to which the UDDI publisher is assigned.

Policy groups:

This topic contains links to the detailed settings information for every policy group that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id*.

To the right of the page is a list of the Policy Groups that can be acted upon. Clicking on a specific group will open the page for the group required.

UDDI keying policy settings:

This topic contains details of the UDDI keying settings that you can configure for a UDDI registry.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Keying policies**.

Registry key generation:

Specifies whether publishers are allowed to publish key generator tModels. You can manage how publishers are allowed to publish key generator tModels by configuring the entitlements in the UDDI Publishers page.

Data type Boolean
Default True (selected)

Registry support of UUID keys:

Specifies whether publisher supplied uuidKeys are allowed in publish requests. You can manage how publishers are allowed to use uuidKeys by configuring the entitlements in the UDDI Publishers page.

Data type Boolean
Default False (cleared)

UDDI node API policy settings:

This topic contains details of the API settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **API policies**.

Note: This information applies only to UDDI Version 3. The settings for Versions 1 and 2 are not changeable; authentication tokens are required for publish requests, but not for inquiry requests (there are no custody transfer requests in Versions 1 or 2).

Authorization for inquiry:

Specifies whether authorization using the authInfo element is required for inquiry API requests. This setting is only relevant if the V3SOAP_Inquiry_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_Inquiry_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true an authorization token will be required to complete the request. If this setting is false, an authorization token is not required; if one is supplied it will be ignored and the request will be processed as if it was made by the default user defined in the UDDI node settings.

Typically, UDDI registries are configured not to require authorization for inquiry API requests.

Data type Boolean
Default False (cleared)

Authorization for publish:

Specifies whether authorization using the authInfo element is required for publish API requests. This setting is only relevant if the V3SOAP_Publish_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_Publish_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true an authorization token will be required to complete the request. If this setting is false, an authorization token is not required; if one is supplied it will be ignored and the request will be processed as if it was made by the default user defined in the UDDI node settings.

Typically, UDDI registries are configured to require authorization for publish API requests.

Data type	Boolean
Default	True (selected)

Authorization for custody transfer:

Specifies whether authorization using the authInfo element is required for custody transfer API requests. This setting is only relevant if the V3SOAP_CustodyTransfer_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_CustodyTransfer_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true an authorization token will be required to complete the request. If this setting is false, an authorization token is not required; if one is supplied it will be ignored and the request will be processed as if it was made by the default user defined in the UDDI node settings.

Typically, UDDI registries are configured to require authorization for custody transfer API requests.

Data type	Boolean
Default	True (selected)

UDDI user policy settings:

This topic contains details of the user policy settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **User policies**.

Allow transfer of ownership:

When true, data ownership can be transferred between owners within the UDDI node.

Data type	Boolean
Default	True (selected)

UDDI data custody policy settings:

This topic contains details of the data custody settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Data custody policies**.

Transfer token expiration period:

Specifies the length of time from the issue of a transfer token, in minutes, after which that token expires.

If you set this value too high, you might expose the UDDI registry to a risk of misuse.

Data type	Integer
Default	1440
Range	1 to 2147483647 (for all intents and purposes, unlimited)

UDDI value set policy:

This topic contains details of the value set policy settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* >**Value set policies**.

Enable checked value sets:

Specifies whether checked value sets are supported. If you set this value to false, publish requests of value set tModels that contain a 'checked' keyedReference will be rejected.

Data type	Boolean
Default	True (selected)

UDDI node miscellaneous:

This topic contains details of the miscellaneous settings that you can configure for a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Miscellaneous policies**.

Node generates discoveryURLs:

Specifies whether a UDDI node can establish a policy on whether it generates discoveryURLs.

Data type	Boolean
Default	False (cleared)

Node supports HTTP Get Service:

Specifies whether the UDDI node supports an HTTP GET service for access to the XML representations of UDDI data structures.

Data type	Boolean
Default	True (selected)

URL prefix for V3 GET servlet:

Specifies the prefix for the URL to the version 3 GET servlet that is used to retrieve the XML representation of a published entity. This property applies to UDDI version 3 API requests only.

The format of the prefix is http://hostname:port/uddiv3soap/, where uddisoap is the context root of the UDDI version 3 SOAP servlet.

When a businessEntity is published, if you have set **Node generates discovery URLs** to true, the discoveryURL value is generated based on this prefix value. Otherwise, the discoveryURL value will be empty.

The UDDI Version 3 specification recommends that you do **not** enable generation of discoveryURLs as they can affect the use of digital signatures. If you do enable generation of discoveryURLs, it is recommended that you do not change the URL prefix afterwards, otherwise discoveryURLs generated using the earlier URL prefix will no longer work.

Data type	URL
Default	http://localhost:9080/uddiv3soap/

Overview of the UDDI registry Administrative Interface

The UDDI registry Administrative Interface allows you to inspect and manage the runtime configuration of a UDDI application. This includes managing the information and the activation state about a UDDI node, updating properties and policies, setting publish tier limits, registration of UDDI publishers, and controlling value set support. The operations of the UDDI registry Administrative Interface can be read and invoked using standard JMX (Java Management Extensions) interfaces.

The use of JMX is explained in Using administrative programs (JMX). See the UDDI registry Administrative Interface for full details on using the UDDI administrative interface.

Backing up and restoring the UDDI registry database

If you want to protect the data in your UDDI registry database, you can back up and restore the database using the facilities of the database product your UDDI node is on.

Cloudscape

To backup a Cloudscape UDDI registry database, first ensure that the UDDI application is stopped (and hence, not accessing the Cloudscape database), and ensure that no other application is using the Cloudscape UDDI30 database. Make a copy of the UDDI30 directory using the file system that the directory resides upon.

To restore the database, replace the UDDI30 file structure with the back up. Note that any updates made since the back up was taken will be lost.

Non-Cloudscape

Use the appropriate backup and restore tools for the database being used to contain the UDDI registry. To use these tools, refer to the documentation for the database product.

Removing and reinstalling the UDDI registry

A UDDI registry node consists of a J2EE application, a store of data (using a relational database management system) referred to as the UDDI database, and a means to connect the application to the data (a data source and related elements). All the data relating to UDDI is stored within the UDDI database and therefore exists irrespective of the UDDI application.

With these facts in mind, consider the following options:

- To remove a node from a WebSphere Application Server you do not have to delete the database. You only have to delete the UDDI application and any associated resources such as the data source used (and J2C Authentication Data if used), as the data in the UDDI database is separate from the UDDI application.
- You do not have to remove the UDDI application to start a new UDDI registry node. Instead you can create a new, replacement node by changing the datasource which the UDDI application uses to access the new UDDI database.

Remove the UDDI application if you no longer want a UDDI facility on a particular WebSphere Application Server (you can subsequently move the UDDI registry node to a different WebSphere Application Server).

Reinstall the UDDI application if you wish to continue to provide the UDDI facility on a particular WebSphere Application Server.

To reinstall a UDDI registry node, see Reinstalling the UDDI application. To remove a UDDI application or to remove a UDDI registry node, see Removing a UDDI Node.

Removing a UDDI registry node

To completely remove a UDDI registry node you need to remove the UDDI registry application and delete the UDDI registry database. However there may be situations where you only want to perform one of these tasks:

Removing the UDDI registry application from an application server.
You might want to do this in order to reinstall the application because it has become corrupted for some reason, or to apply service. See also the topic on Reinstalling the UDDI registry application.

Deleting the UDDI registry database
You might want to do this in order to use a different database product as the persistence store for your UDDI data, to delete all your UDDI registry data in order to publish fresh data (for example, if you have completed a test cycle), or to cause the UDDI registry node to re-initialize with new UDDI property settings (for example, to move from a default UDDI node to a customized UDDI node). Note that deleting a UDDI registry database will cause all UDDI data for that UDDI registry to be lost.

Moving a UDDI registry to another server or profile
You might want to do this if you have created a new profile and want to move the UDDI registry to it.

Completely removing a UDDI registry node from an application server
You might want to do this to remove a UDDI registry that has been used for testing.

Depending on what you wish to achieve, complete **one** of the following steps:

1. **Removing a UDDI registry application**

   To remove the UDDI application from an application server, run the wsadmin script uddiRemove.jacl, as shown, from the *app_server_root*/bin directory.

   The syntax of the command is as follows (this is a Windows platform example; for UNIX or Linux platforms, add the .sh suffix to the wsadmin command):

   ```
   wsadmin [-profileName profile_name] -f uddiRemove.jacl
                  node_name
                  server_name
                  [default]
   ```

   where
   - '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
   - *node_name* and *server_name* are the names of the WebSphere node and application server in which the UDDI application is deployed (these are the names that you specified when you ran uddiDeploy.jacl to install the UDDI application).
   - 'default' is optional and is applicable only for Cloudscape, and then only if the default option was used when the uddiDeploy.jacl script was run to deploy the UDDI registry. Specifying default will remove the UDDI Cloudscape datasource but **not** the UDDI Cloudscape database.

   Output will appear on the screen by default. To direct the output to a log file, add '> *removeuddi.log*' (where removeuddi.log can be any log name of your choice) to the end of the command.

   For example, to remove the UDDI application from server 'server1' running in node 'MyNode' on a Windows system, and send any messages to the file removeuddi.log:

   ```
   wsadmin -f uddiRemove.jacl MyNode server1 > removeuddi.log
   ```

   Note: You can also remove the UDDI registry application using the administrative console in the usual way, by selecting the application in the **Enterprise Applications** view and clicking **Uninstall**.

2. **Deleting a UDDI registry database**

   Note that this will cause all UDDI data in that UDDI registry to be destroyed.

   a. Stop the server that is hosting the UDDI registry application.

b. Delete the database:
   - For DB2, use the database facilities to delete the UDDI database.
   - For Oracle, delete the schemas IBMUDDI, IBMUDI30 and IBMUDS30.
   - For Cloudscape, delete the directory tree containing the UDDI database. By default, this will be located in *app_server_root*/profiles/*profile_name*/databases/com.ibm.uddi/UDDI30.

3. **Moving a UDDI registry to another server or profile**
   a. Make sure that the UDDI database will still be accessible. For example if you are using a remote database make sure that the new server can access it. If your database will not be accessible, or it will be deleted after the move, copy it to a new, accessible location. For example, if you want to move the UDDI registry to a new profile and then delete the old profile, any databases stored in the profile (such as a Cloudscape database created as part of the creation of a default UDDI node) will also be deleted.
   b. Remove the UDDI registry application by running the uddiRemove.jacl script as described above.
   c. If you ran the uddiRemove.jacl script using the default option, the datasource and related objects have already been deleted. If the default option was not valid for your configuration, delete the following objects:
      - the UDDI datasource that references the UDDI registry database (this was created when you set up the UDDI registry).
      - any UDDI JDBC provider that was created (if you did not reuse an existing JDBC provider).
      - any J2C Authentication Data Entry that was created.
   d. In the new server create a J2C authentication data entry (if appropriate), JDBC provider and datasource, to reference the existing database (for more information see the relevant steps in Setting up a customized UDDI node).
   e. Deploy the UDDI registry application as described in Deploying the UDDI registry application, noting that you should not specify the default option even if you previously used this option to set up a default UDDI node. If you use the default option, you might encounter an error during deployment, or in some circumstances your existing UDDI data might be overwritten.

      Note: The UDDI node name will remain the same as before. If the UDDI node name included the node name and server name of the original server, there will be a mismatch between the UDDI node name and the node name and server name of the new server. However this name mismatch will not affect the functioning of the UDDI registry node.
   f. Check that the UDDI data can be accessed. If you are using a copy of the original UDDI registry database, you can now delete the original as described above.

4. **Completely removing a UDDI registry node**

   To completely remove a UDDI registry node from an application server, you need to remove the UDDI registry application and database, and also the resources that were used to reference the UDDI registry database.
   a. Remove the UDDI registry application by running the uddiRemove.jacl script as described above.
   b. Delete the UDDI registry database, as described above.
   c. If you ran the uddiRemove.jacl script using the default option, the datasource and related objects have already been deleted so no further action is required. If the default option was not valid for your configuration, delete the following objects:
      - the UDDI datasource that references the UDDI registry database (this was created when you set up the UDDI registry).
      - any UDDI JDBC provider that was created (if you did not reuse an existing JDBC provider).
      - any J2C Authentication Data Entry that was created.

Reinstalling the UDDI registry application

Perform this task if you want to continue providing UDDI services with your existing UDDI database, but require that the UDDI application code be changed.

1. Make a note of any changes that you have made to the installed UDDI application that you wish to keep, for example changes to security role mappings, changes to the deployment descriptor (web.xml) in v3soap,war, v3gui.war, v3soap.war, or soap.war, or customization of the UDDI user interface (GUI). All such changes will be lost during the reinstallation process; any changes that you wish to keep will need to be reapplied later.
2. Run the wsadmin script uddiDeploy.jacl from the *app_server_root*/bin directory, as shown, noting that:
   - you should not specify the default option even if you previously used this option to set up a default UDDI node using Cloudscape. If you use the default option, you might encounter an error during deployment, or in some circumstances your existing UDDI data might be overwritten.
   - if you are deploying the UDDI registry into a network deployment scenario, ensure that the deployment manager is the target.

   At a command prompt enter:

   ```
   wsadmin [-conntype none] [-profileName profile_name] –f uddiDeploy.jacl
                node_name
                server_name
   ```

   where
   - '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
   - '-conntype none' is optional, and is only needed if the application server or deployment manger is not running.
   - *node_name* and *server_name* are the names of the WebSphere node and application server in which the UDDI application is deployed (these are the names that you specified when you ran uddiDeploy.jacl to install the UDDI application).

   The output from this command can optionally be redirected to a log file by adding '> *log_name*.log' at the end of the command, where *log_name*.log is the name of the log file to be created.

   The command removes the existing UDDI application and reinstalls it.

   Note: Your existing JDBC provider, datasource and any J2C authdata entry will be unchanged by this procedure . Your existing UDDI registry data, including UDDI entities as well as property and policy settings, will also be unaffected.
3. If you noted any changes in step 1, reapply them now.
4. Start or restart the application server for the reapplied changes to take effect.

Applying an upgrade to the UDDI registry

Perform this task to apply an iFix, Fix Pack or Refresh Pack to the UDDI registry.

Note: Some upgrades may require additional steps; refer to the readme file for the upgrade before you complete this task.

1. Apply the WebSphere Application Server iFix, Fix Pack or Refresh Pack to your application server or servers using the WebSphere Application Server Update Installer. Note that this process must be repeated for each server into which you choose to incorporate the UDDI upgrade.
2. If you have not yet deployed a UDDI registry, no further action is required as updates to the UDDI registry will take effect when you first deploy UDDI into any of your application server profiles.
3. If you have already deployed a UDDI registry to one or more application server profiles, redeploy the UDDI application as described in Reinstalling the UDDI registry application, to apply the upgrade. The existing UDDI application will be removed and the updated application will be deployed.

Configuring the UDDI registry application

The UDDI registry is supplied as a Java 2 Platform, Enterprise Edition (J2EE) application file, uddi.ear.

You can configure the following aspects of the UDDI registry:
- Configuring UDDI registry security
- Configuring SOAP API and GUI services
- Multiple language encoding support in UDDI
- Customizing the UDDI registry user interface (GUI)

Configuring UDDI registry security

The UDDI Version 3 registry is designed to exploit the advantages of WebSphere Application Server security. The registry also supports the UDDI Version 1 and Version 2 security features and the UDDI Version 3 Security API.

For production use, it is recommended that the UDDI Version 3 registry is configured to use WebSphere Application Server security, and that use of UDDI Version 1 and Version 2 security features and the Version 3 Security API is avoided. However, for solutions with a strong preference for UDDI security, the UDDI Version 3 registry can be configured to enable this, both when WebSphere Application Server security is enabled and when it is disabled.

To configure UDDI registry security, complete the following steps:
1. Follow the appropriate link for the type of configuration you wish to set up:
   - "Configuring the UDDI registry to use WebSphere Application Server security" on page 507
   - "Configuring the UDDI registry to use UDDI security" on page 508
2. Review the "UDDI registry security additional considerations" on page 510.

Configuring the UDDI registry to use WebSphere Application Server security

Before starting this task complete the following two steps:
- Enable WebSphere Application Server security (see Enabling security for all application servers). This will allow the UDDI registry to exploit the WebSphere Application Server security features.
- Ensure that WebSphere Application Server is configured to use HTTPS (SSL); this will allow the use of secure access with the UDDI registry. WebSphere Application Server is configured by default to accept SSL requests on port 9443, however if you need to make any additional SSL configuration changes, refer to SSL configurations for selected scopes.

There are two aspects of WebSphere Application Server security which are exploited by the UDDI registry:

Authorization
: Authorization determines whether users are allowed access to services. WebSphere Application Server determines authorization by mapping users, or groups of users, to roles. UDDI makes use of two WebSphere Application Server special subjects: *Everyone* (all users are allowed access) and *AllAuthenticatedUsers* (only valid WebSphere Application Server registered users are allowed access).

Data confidentiality
: Data confidentiality determines security at the transport level. Data confidentiality for WebSphere Application Server services can be either 'none' (HTTP is used as the transport protocol) or 'confidential' (requiring the use of SSL; HTTPS is used as the transport protocol).

When WebSphere Application Server security is enabled, the default settings in the UDDI Version 3 Application and Web deployment descriptors result in the following features:

- Publish, Custody Transfer and Security services are mapped to the AllAuthenticatedUsers special subject, and data confidentiality is enforced (HTTPS is used). Authentication uses the standard WebSphere Application Server security facilities and there is no separate registration function for the UDDI registry. To use publish functions, users must supply their WebSphere Application Server user name and password (unless you have modified the supplied publish role), and must also be registered UDDI Publishers. By registering users as UDDI Publishers you control which users in the AllAuthenticatedUsers subject can update the UDDI registry.
- Inquiry services are mapped to the Everyone special subject, and data confidentiality is not enforced (HTTP is used). To use inquiry services, users do not need to supply a user name or password, and do not have to be registered UDDI publishers.

You are recommended to use the default settings as described above. However, you can change the defaults by mapping roles to different users or user groups, or by not mapping a role to any users or user groups, in which case all access to that role will be disabled. If you do map roles to users or groups, turn on the **Automatically register UDDI publishers** property (see UDDI node settings) so that you do not have to use two mechanisms for giving access to a subset of users.

For more information about UDDI role mappings, and a list of UDDI registry services and roles, see Access control for UDDI registry interfaces.

To change the default settings follow the steps below:

1. To change the role mappings using the administrative console, complete the following steps:
   a. In the navigation pane, click **Applications** → **Enterprise Applications**.
   b. In the content pane, click the UDDI registry application.
   c. Under **Detail Properties** click **Security role to user/group mapping**.
   d. Make any changes you require and click **OK**.
2. To change the role mappings using the wsadmin command, complete the following steps:
   a. Use the MapRolesToUsers option of the edit command of the AdminApp object to map the roles defined in the UDDI registry application to special subjects (Everyone or AllAuthenticatedUsers), to users, or to user groups. For example, the following Java command language (JACL) statement maps the Version 3 GUI Publish role to Everyone, and the Version 3 SOAP Publish role to user 'user1' and group 'group1':

      ```
      $AdminApp edit $AppName {-MapRolesToUsers { {"GUI_Publish_User" Yes No "" ""} {"V3SOAP_Publish_
      User_Role" No No "user1" "group1"} }}
      ```

      where $AppName is a variable representing the name of the UDDI registry application.

      For more information about using the MapRolesToUsers option, see Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands.
3. To change the data confidentiality settings, refer to "Configuring SOAP API and GUI services" on page 512.

Configuring the UDDI registry to use UDDI security

It is possible to exploit the UDDI registry security features when WebSphere Application Server security is either enabled or disabled. Each situation requires different configuration, and different behavior is achieved.

Note: While useful for test purposes, it is not anticipated that WebSphere Application Server security is disabled for production configurations.

To continue configuring the UDDI registry to use UDDI security, choose one of the following options:

- "Configuring UDDI Security with WebSphere Application Server security enabled" on page 508
- "Configuring UDDI Security with WebSphere Application Server security disabled" on page 509

Configuring UDDI Security with WebSphere Application Server security enabled:

When WebSphere Application Server security is enabled, to use the UDDI Version 1 and Version 2 publish security features (use of authentication tokens) or the UDDI Version 3 security API, use the administrative console to complete the following steps:

1. In the navigation pane, click **Applications** → **Enterprise Applications**.
2. In the content pane, click the UDDI registry application. Under **Detail Properties** click **Security role to user/group mapping**.
3. Set the WebSphere Application Server security role mappings to Everyone for the following UDDI services:
   - Versions 1 and 2 SOAP publish service (SOAP_Publish _User)
   - Version 3 publish service (V3SOAP_Publish_User_Role)
   - Version 3 custody transfer service (V3SOAP_CustodyTransfer_User_Role)
   - Version 3 security service (V3SOAP_Security_User_Role)

   Changing the role mappings to Everyone prevents WebSphere Application Server security from overriding UDDI security.
4. Ensure that UDDI Policy is set to require the use of authentication tokens for the UDDI Version 3 Publish and Custody Transfer services (use of authentication tokens is already required for Version 1 and Version 2 Publish services). To do this, click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups** click **API policies**. Select the **Authorization for publish** and **Authorization for custody transfer** check boxes. (Select the **Authorization for inquiry** check box if you require authentication for UDDI Inquiry services).
5. Click **OK**.

With this configuration, no Security Role authentication restriction is imposed, but the credentials (user name and password) associated with the authentication token are authenticated by WebSphere Application Server.

Note: When WebSphere Application Server security is enabled, WebSphere Application Server data confidentiality management is independent of UDDI security and is managed as described in "Configuring the UDDI registry to use WebSphere Application Server security" on page 507.

Configuring UDDI Security with WebSphere Application Server security disabled:

With WebSphere Application Server security disabled, neither WebSphere Application Server security roles nor data confidentiality constraints apply. This mode may be useful for test UDDI registry configurations.

In this mode, UDDI Version 1 and Version 2 security features are active:

- UDDI Version 1 and Version 2 publish requests require UDDI Version 1 and Version 2 authentication tokens respectively. Publishers requesting or using an authentication token must be registered WebSphere Application Server users.
- UDDI Version 1 and Version 2 inquiry requests do not require authentication tokens.

No further configuration is required for UDDI Version 1 and Version 2 security.

For UDDI Version 3, the use of the UDDI Version 3 Security API, and the use of authentication tokens with Version 3 Publish and Custody Transfer APIs, is optional. To make use of these UDDI Version 3 security features, use the administrative console to complete the following steps:

1. Specify that use of authInfo is required. Click **UDDI** → **UDDI Nodes** > *uddi_node_name*.
2. In the **General Properties** section, select the **Use authInfo credentials if provided** check box.
3. Click **OK**.

Authentication tokens will now be required for publish and custody transfer requests, but not for inquiry requests. Publishers requesting or using an authentication token must be registered WebSphere Application Server users.

Access control for UDDI registry interfaces

Access to UDDI registry interfaces is controlled by a combination of J2EE declarative security using role mappings, and UDDI properties and policies such as the registering of users as UDDI publishers.

Each of the UDDI registry interfaces is represented by a security role. The interfaces and their corresponding roles are as follows:

UDDI registry interface	Security role
Version 3 SOAP inquiry	V3SOAP_Inquiry_User_Role
Version 3 SOAP publish	V3SOAP_Publish_User_Role
Version 3 SOAP custody transfer	V3SOAP_CustodyTransfer_User_Role
Version 3 SOAP security	V3SOAP_Security_User_Role
Version 3 GUI inquiry	GUI_Inquiry_User
Version 3 GUI publish	GUI_Publish_User
Versions 1 and 2 SOAP inquiry	SOAP_Inquiry_User
Versions 1 and 2 SOAP publish	SOAP_Publish_User
EJB inquiry	EJB_Inquiry_Role
EJB publish	EJB_Publish_Role

By default, the inquiry roles are mapped to the *Everyone* special subject and the non inquiry roles are mapped to the *AllAuthenticatedUsers* special subject. For more information about WebSphere Application Server role mapping and the Everyone and AllAuthenticatedUsers special subjects, see Role-based authorization. With these default settings, after you enable WebSphere Application Server security you do not need access control to use the UDDI registry inquiry interfaces, however to use the publish roles and the Version 3 custody transfer role you must be authenticated using a WebSphere Application Server userid and password. (The Version 3 security role is a special case, as this concerns use of UDDI registry security instead of WebSphere Application Server security, and must be specially configured as described in Configuring the UDDI registry to use UDDI security.)

For more information about UDDI registry security roles and how they can be used to control authorization and data confidentiality, see Configuring the UDDI registry to use WebSphere Application Server security.

Roles which are mapped to AllAuthenticatedUsers (as the UDDI registry publish interfaces are by default) are further protected in that, having successfully authenticated, the user must also be registered as a UDDI publisher in order to publish data to the UDDI registry. An E_unknownUser error is returned in the disposition report if the user is not registered. You can register users as UDDI publishers in one of two ways:

- Create a new UDDI publisher using the administrative console (see UDDI Publisher collection) or the JMX interface (see UDDI registry Administrative (JMX) Interface).
- Set the **Automatically register UDDI publishers** property (see UDDI node settings), in which case users will be automatically registered as a publisher on their first publish request.

In accordance with the UDDI specification, there is additional access control in that an entity which has been published to the UDDI registry can only be updated or deleted by the user who originally published that entity.

The UDDI registry also provides some management interfaces which are protected by requiring administrative permissions for certain operations; see UDDI registry Management Interfaces for details.

UDDI registry security additional considerations

In addition to the configuration of UDDI registry security, there a number of other UDDI registry settings which may affect the behavior of the UDDI registry. Some of these settings are security specific, others are points to bear in mind when configuring security.

Additional security considerations

UDDI registry interfaces are protected as detailed in Access control for UDDI registry interfaces.

The UDDI registry supports the use of XML Digital Signatures to sign UDDI entities. This is described in Use of digital signatures with the UDDI registry.

Additional policy considerations

A number of the UDDI property and policy settings also determine the behavior of a UDDI registry with respect to security.

To review or change the following property settings, click **UDDI** → **UDDI Nodes** > *uddi_node_name*. The settings are also detailed in the administrative console help.

Key space requests require digital signature
: This setting determines whether all tModel:keyGenerator requests for key space must be digitally signed. To understand key space refer to UDDI registry Version 3 Entity Keys.

Use authInfo credentials if provided
: This setting applies only when WebSphere Application Server security is disabled. See Configuring UDDI Security with WebSphere Application Server security disabled.

Authentication token expiry period
: The authentication token expiry period is the length of idle time (in minutes) allowed before an authentication token becomes invalid.

Default user name
: The default user name is used for publish operations when WebSphere Application Server security is disabled and no authentication token data is supplied.

To review or change the following policy settings, click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups**, click **API policies**. The settings are also detailed in the administrative console help.

Authorization for inquiry
: Specifies whether authorization using authentication tokens is required for inquiry API requests.

Authorization for publish
: Specifies whether authorization using authentication tokens is required for publish API requests.

Authorization for custody transfer
: Specifies whether authorization using authentication tokens is required for custody transfer API requests.

The above policy settings apply when UDDI security features are being used and WebSphere Application Server security is enabled. If the UDDI service in question is mapped to the security role AllAuthenticatedUsers, these settings will be overridden. See Configuring UDDI Security with WebSphere Application Server security enabled.

Other considerations

The publish related actions that a registered UDDI publisher can perform are defined by their entitlements, as described in UDDI registry user entitlements.

In addition to the property and policy settings above, be aware that some UDDI keying and user policy settings also influence publish behavior. These settings are not specific to security, but you should bear them in mind as they also place restrictions on successful completion of publish requests.

To review or change the following property settings, click **UDDI** → **UDDI Nodes** > *uddi_node_name*. The settings are also detailed in the administrative console help

Automatically register UDDI publishers
The UDDI registry requires publisher entitlements to be set before allowing any publish requests. This option automatically registers users with default entitlements.

If this option is not selected, users (and their entitlements) can be registered. See UDDI Publisher settings.

Use tier limits
If selected, tier limits are enforced.

If this option is selected you should have one or more tiers configured (see Tier collection and UDDI Tier settings). You should also ensure that registered UDDI Publishers are assigned to a tier (see UDDI Publisher settings).

To review or change the following property setting, click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups** click **Keying policies**. The setting is also detailed in the administrative console help.

Registry key generation
If this option is selected, publishers may request key space and, if successful, publish with publisher assigned keys.

UDDI registry user entitlements: UDDI registry *user entitlements* define the set of publish related actions that registered UDDI users are entitled to perform.

An important entitlement is the number of entities of each type that a UDDI user is entitled to publish; this is controlled by assigning the user to a UDDI publisher *tier*. Any number of tiers can be defined to the UDDI registry, and there are some predefined tiers which are supplied when the UDDI registry is deployed. A UDDI publisher tier specifies the maximum number of each entity (business, service, binding, tModel, and publisher assertion) that a user assigned to that tier may publish. (See UDDI Tier settings for information about defining UDDI publisher tiers.)

Other entitlements relate to the user's entitlement to allocate key spaces within which they can specify publisher assigned keys when publishing UDDI entities. A key space is allocated by publishing a keyGenerator tModel, and there are a number of entitlements relating to different kinds of key generator. For full details of these entitlements, see UDDI Publisher settings. For more information about key generators and publisher assigned keys, see UDDI registry Version 3 Entity Keys.

The entitlements for a UDDI user can be set using the administrative console, or through JMX using the UDDI Administrative interface, as described in UDDI node collection and UDDI registry Administrative (JMX) Interface.

Configuring SOAP API and GUI services

Configuring Version 1 and Version 2 SOAP API services

You can configure the following Version 1 and Version 2 SOAP interface properties:

- *defaultPoolSize*. This is the number of SOAP parsers with which to initialize the parser pool for the SOAP interface. You can set this independently for the Publish (uddipublish) and Inquiry (uddi) APIs. For example, if you expect more inquiries than publish requests through the SOAP interface, you can set a larger pool size for the Inquiry API. The default initial size for both APIs is 10.

- Whether the API is to be secure (accessed using HTTPS) or insecure (accessed using HTTP). The default for Publish is to use HTTPS and Inquiry to use HTTP.

To configure these properties after the UDDI application has been installed:

1. Edit the active deployment descriptor (web.xml) for the Version 1 and Version 2 SOAP module (soap.war).

   This file is located in the following directory:

   app_server_root/profiles/*profile_name*/config/cells/*cell_name*/applications/
   UDDIRegistry.*node_name*.*server_name*.ear/deployments/
   UDDIRegistry.*node_name*.*server_name*/soap.war/WEB-INF
2. To modify defaultPoolSize for Version 1 or Version 2 Publish, modify the 'param-value' element in the servlet with 'servlet-name' = uddipublish
3. To modify defaultPoolSize for Version 1 or Version 2 Inquiry, modify the 'param-value' element in the servlet with 'servlet-name' = uddi
4. To modify the user data constraint transport guarantee for Version 1 or Version 2 publish, which determines whether the Publish service is to be confidential (accessed using HTTPS) or insecure (using HTTP), find the 'security-constraint' with id = 'UDDIPublishTransportConstraint' and set its 'user-data-constraint' 'transport-guarantee' to CONFIDENTIAL or NONE.
5. Stop and restart the application server for the changes to take effect.

Configuring Version 3 SOAP API services

For the Version 3 SOAP interface, you can specify whether the Publish, Custody Transfer, Security and Inquiry API services are to be secure (accessed using HTTPS) or insecure (accessed using HTTP). The default for Publish, Custody Transfer and Security APIs is to use HTTPS, and Inquiry to use HTTP.

To configure these properties after the UDDI application has been installed:

1. Edit the active deployment descriptor (web.xml) for the Version 3 SOAP module (v3soap.war).

   This file is located in the following directory:

   app_server_root/profiles/*profile_name*/config/cells/*cell_name*/applications/
   UDDIRegistry.*node_name*.*server_name*.ear/deployments/
   UDDIRegistry.*node_name*.*server_name*/v3soap.war/WEB-INF
2. Each type of service is represented in the deployment descriptor (web.xml) file by a <security-constraint> element, which contains, amongst other things, a <display-name> element and a <user-data-constraint> <transport-guarantee> element. Locate the <security-constraint> element for the service that you want to modify by searching the deployment descriptor (web.xml) file for the relevant <display-name> value (see the table below). When you have found the <security-constraint> element, set its <user-data-constraint> <transport-guarantee> to either CONFIDENTIAL (the service can only be accessed using HTTPS) or NONE (the service can be accessed using HTTP).

Type of UDDI service	Value of <security-constraint> <display-name> element
Publish	AxisServlet Publish Resource Collection
Custody transfer	AxisServlet CustodyTransfer Resource Collection
Security	AxisServlet Security Resource Collection
Inquiry	AxisServlet Inquiry Resource Collection

3. Stop and restart the application server for the changes to take effect.

Configuring Version 3 GUI services

For the Version 3 GUI interface, you can specify whether the Publish and Inquiry API services are to be secure (accessed using HTTPS) or insecure (accessed using HTTP). The default for Publish is to use HTTPS, and Inquiry to use HTTP.

To configure these properties after the UDDI application has been installed:

1. Edit the active deployment descriptor (web.xml) for the Version 3 GUI module (v3gui.war).

   This file is located in the following directory:

   ```
   app_server_root/profiles/profile_name/config/cells/cell_name/applications/
     UDDIRegistry.node_name.server_name.ear/deployments/
       UDDIRegistry.node_name.server_name/v3gui.war/WEB-INF
   ```

2. Use the table below to find the `id` value for the `<user-data-constraint>` element that corresponds with the service that you want to change. Search for this value in the deployment descriptor (web.xml) file to find the `<user-data-constraint>` element, then set the `<user-data-constraint> <transport-guarantee>` to either CONFIDENTIAL (the service can only be accessed using HTTPS) or NONE (the service can be accessed using HTTP).

Type of UDDI service	Value of <user-data-constraint id>
Publish	UDDIPublishTransportConstraint
Inquiry	UDDIInquireTransportConstraint

3. Stop and restart the application server for the changes to take effect.

Multiple language encoding support in UDDI

UDDI API

UDDI Version 3 supports both UTF-8 and UTF-16 encoding. Internally UTF-16 characters are stored as UTF-8. This is transparent to the user application.

UDDI User Console

The UDDI user console only supports UTF-8 encoding. To enable this, you must configure the application server into which the UDDI registry application is installed with UTF-8 encoding enabled. To do this, refer to ″Configuring application servers for UTF-8 encoding″ elsewhere in the WebSphere Information Center.

Customizing the UDDI registry user interface (GUI)

The look and feel of the UDDI registry user interface is determined by the styles defined in the *.css* files located in the following directory: *app_server_root*/profiles/*profile_name*/installedApps/*cell_name*/UDDIRegistry.*node_name*.*server_name*.ear/v3gui.war/theme.

Style class definitions in these files can be edited to alter the overall theme of the UDDI registry user interface, including font attributes, layout and colors.

Managing the UDDI registry

You can use either the WebSphere Application Server administrative console or the Java Management Extensions (JMX) management interface to manage UDDI Registries.

In previous versions of WebSphere Application Server and the UDDI registry, a properties file was used, but from WebSphere Application Server Version 6, you manage all the policies and properties of the UDDI registry through either the JMX management interface or the administrative console.

JMX can be used to monitor and configure UDDI registries programmatically, and is explained in Using administrative programs (JMX). See UDDI registry Administrative (JMX) Interface for full details on using the UDDI administrative interface. To manage UDDI registries using the WebSphere Application Server administrative console, start from the UDDI link in the left navigation pane as described below.

Using the UDDI management functions available in the WebSphere Application Server administrative console, you can perform the following operations:

- view and manage the status of all UDDI nodes in a cell
- initialize UDDI nodes with required settings
- configure general properties that affect UDDI runtime behavior
- manage UDDI policy settings
- create, view and update UDDI publishers
- create, view and update publisher tiers which limit how many UDDI entries may be published
- view and manage the status of value sets

If WebSphere Application Server security is enabled, you will need to log in to the WebSphere Administrative Console, supplying a valid userid and password, in order to use the UDDI management functions.

UDDI node collection

Use this page to manage the UDDI nodes in this cell. Each UDDI node represents an individual UDDI registry application. A UDDI node will only appear in this list if its underlying UDDI application is started. The status of the UDDI node indicates whether the node is activated (available to accept API requests), deactivated (not allowing user requests), or not initialized. UDDI nodes that are not initialized require some properties to be set before they can be initialized and activated.

To view this administrative console page, click **UDDI** → **UDDI Nodes**.

Each UDDI node is represented by a UDDI Node ID, Description, UDDI Application Location and Status. To manage an individual UDDI node, click on its UDDI Node ID link. This takes you to the Configuration page where you can manage its general properties, initialize it if the status is set to *Initialization Pending*, and access pages for managing policies, UDDI publishers, tiers and value sets.

UDDI Node ID

Specifies the identifier for the UDDI node.

Description

Specifies the description of the UDDI node.

UDDI Application Location

Specifies the server in which the UDDI registry application is deployed and running.

For a cluster configuration, which will include several servers all running the UDDI registry application, you will see a single location listed; this location could represent any of the active servers.

Status

Specifies the status of the UDDI node.

The Status of the UDDI node can be *Not initialized*, *Initialization pending*, *Initialization in progress*, *Migration in progress*, *Migration pending*, *Value set creation in progress*, *Value set creation pending*, *Activated* or *Deactivated*. If a node is in the *Initialization pending* state, you must initialize it before you can activate it. If you attempt to initialize the node and it remains in a pending state, an error occurred during migration or initialization.

To activate UDDI nodes that are *Deactivated*, select them by checking the corresponding check boxes in the Select column and click **Activate**. Similarly to deactivate UDDI nodes, select them and click **Deactivate**.

Note: Restarting the UDDI application, or the application server, will always result in the reactivation of the UDDI node, even if the node was previously deactivated.

UDDI node settings

This topic contains details of the general properties that you can configure for a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id*.

By clicking on a node in the UDDI node ID column the UDDI node detail page is displayed. The UDDI node detail page displays a set of General Properties for the UDDI node, some of which may be editable depending on the state of the node. There are also links to Additional Properties (Value sets, Tiers and UDDI Publishers) and links to Policy Groups where you can view and change UDDI node policy.

Unless you installed the UDDI node as a default UDDI node (as defined in UDDI registry terminology), there are some important general properties that you must set before you can initialize the UDDI node. These properties are marked as being required (indicated by the presence of a '*' next to the input field). You may set the values as many times as you wish before initialization. However, after you initialize the UDDI node, these properties will become read only for the lifetime of that UDDI node. It is very important to set these properties correctly. You can set other general properties of the UDDI node before, and after, initialization.

After you set the general properties to appropriate values, click **OK** to save your changes and exit the page, or **Apply** to save your changes and remain on the same page. At this point the changes are stored.

If the UDDI node is in the *Not Initialized* state, indicated on the UDDI node detail page by the presence of an **Initialize** button (above the General Properties section), you can initialize UDDI node by clicking **Initialize**. This operation can take a while to complete. It is important to save any changes you made to the general properties by clicking **Apply** or **OK**, before you click **Initialize**.

UDDI Node ID:

Specifies the unique identifier that is given to a UDDI node in a UDDI registry. The node ID must be a valid UDDI key. The value will also be the domain key for the UDDI node.

The value for the node ID will also be the domain key for this UDDI node.

Required	Yes
Data type	String
Default	uddi:*cell_name*:*node_name*:*server_name*:node_id

UDDI node description:

Specifies the description of this UDDI node.

Required	Yes
Data type	String
Default	WebSphere UDDI registry default node

Root key generator:

Specifies the root key space of the registry. Registries intending to become affiliate registries might want to specify a root key space in a partition below the root key generator of the parent root registry, for example, uddi:thisregistry.com:keygenerator.

Required	Yes
Data type	String
Default	uddi:*cell_name*:*node_name*:*server_name*:keyspace_id:keygenerator

Prefix for generated discoveryURLs:

Specifies the URL prefix that is applied to generated discoveryURLs in businessEntity elements, so they can be returned on HTTP GET requests. This property applies to UDDI version 2 API requests only. Set this prefix to a valid URL for your configuration, and do not change it unless absolutely necessary.

The format is `http://hostname:port/uddisoap/`, where `uddisoap` is the context root of the UDDI version 2 SOAP servlet.

Although this field is not required, you should set it so that the required and valid URL is generated in response to version 2 GET requests. After you have set the prefix you should not alter it unless a subsequent configuration change renders it invalid; if you change the prefix, discoveryURLs generated using the earlier prefix will no longer work.

Required	No
Data type	String
Default	http://localhost:9080/uddisoap

Host name for UDDI node services:

Specifies the host name root used by the UDDI node to model API services in its own node business entity. This value must be the fully qualified domain name, or IP address, of the network host.

The UDDI node provides Web services that implement each of the UDDI API sets that it supports. The host name is used to generate access point URLs in the bindingTemplate elements for each of the services. The access point URL is generated by prefixing the host name value with a protocol, such as http, and suffixing it with the corresponding host port number. The access point URL must resolve to a valid URL.

Data type	String
Default	localhost

Host HTTP port:

Specifies the port number used to access UDDI node services with HTTP. This port number must match the WebSphere Application Server port for HTTP requests.

Data type	Integer
Default	9080

Host HTTPS port:

Specifies the port number used to access UDDI node services with HTTPS. This port number must match the WebSphere Application Server port for HTTPS requests.

Data type	Integer

Default 9443

Maximum inquiry result set size:

Specifies the maximum size of result set which the registry will process for an inquiry API request.

If the result set exceeds this value, an E_resultSetTooLarge error is returned to the user. If you set this value too low, and users use imprecise search criteria, the likelihood of receiving an E_resultSetTooLarge error is increased. Setting the value higher allows larger result sets but can cause increased response times.

Data type Integer
Default 500
Range 0 to 1024

Maximum inquiry response set size:

Specifies, for inquiry API requests, the maximum number of results returned in each response. Do not set this value higher than the value for **Maximum inquiry results**.

If the result set contains more results than this value, the response will include only a subset of those results. The user can retrieve the remaining results using the listDescription feature as described in the UDDI specification. If you set this value too low, the user has to make more requests to retrieve the remainder of the result set.

Data type Integer
Default 500
Range 0 to 1024

Maximum search names:

Specifies the maximum number of names that can be supplied in an inquiry API request. To avoid slowing UDDI node response times, set this value no higher than 8.

Higher values allow more complex requests to be processed by the UDDI node, however complex requests can significantly slow the response times of the UDDI node.

Data type Integer
Default 5
Range 1 to 64

Maximum search keys:

Specifies the maximum number of keys that can be supplied in an inquiry API request. To avoid slowing UDDI node response times, set this value no higher than 5.

This value limits the number of references that can be specified in categoryBag, identifierBag, tModelBag and discoveryURLs.

Higher values allow the UDDI node to process more complex requests, however complex requests can significantly slow the response times of the UDDI node. In exceptional cases, the UDDI node may reject complex requests with excessive numbers of keys even if the value of maxSearchKeys is not exceeded.

Data type	Integer
Default	5
Range	1 to 64

Key space requests require digital signature:

Specifies whether tModel:keyGenerator requests must be digitally signed.

Data type	Boolean (check box)
Default	False (cleared)

Use tier limits:

Specifies whether an approval manager is used to check publication tier limits. If you set this value to false, the number of UDDI entities that can be published is unlimited.

Data type	Boolean (check box)
Default	True (selected)

Use authInfo credentials if provided:

Specifies whether authInfo contents in UDDI API requests are used to validate users when WebSphere Application Server security is off. If you set this value to true, the UDDI node will use the authInfo element in the request, otherwise the default user name is used.

Data type	Boolean (check box)
Default	True (selected)

Authentication token expiry period:

Specifies the period, in minutes, after which an authentication token is invalidated and a new authentication token is required.

Set this value high enough to allow the registry to operate successfully, but be aware that high values can increase the risk of illegal use of authentication tokens.

Data type	Integer
Default	30
Range	1 to 10080 minutes (10080 minutes = 1 week)

Automatically register UDDI publishers:

Specifies whether UDDI publishers are automatically registered and assigned to the default tier. Automatically registered UDDI publishers are given default entitlements.

Data type	Boolean (check box)
Default	True (selected)

Default user name:

Specifies the user name used for publish operations when WebSphere Application Server security is disabled and **Use authInfo credentials if provided** is set to false.

Data type	String
Default	UNAUTHENTICATED

Default language code:

Specifies, for UDDI version 1 and version 2 requests, the default language code to be used for xml:lang, when not otherwise specified.

Data type	String
Default	en

Value set collection:

Use this page to view and configure the value sets that are installed in a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Value Sets**.

Value sets in a UDDI node are either supported or not supported by policy. By default, new value sets are *not* supported. After you publish a value set tModel and loaded value set data, you can control whether other UDDI entities can reference this value set tModel by setting the *Supported* policy.

To enable support for one or more value sets, select the value sets by selecting the appropriate check boxes in the **Select** column. Click **Enable Support**. The Supported field for all the selected value sets is updated, with a value of true, to reflect the new status.

To disable support for a value set, which might be necessary before you remove it from the UDDI node, select the value sets in the same manner as for enabling support. Click **Disable Support**. The Supported field for all the selected value sets is updated, with a value of false, to reflect the new status.

Clicking on a value set name in the list takes you to the general properties page for that value set as described in Value set settings.

Name:

Specifies the name of the tModel that represents this value set.

tModelkey:

Specifies the key for the tModel that represents this value set.

Supported:

Specifies whether this value set is supported by policy in this UDDI node.

Value set settings:

Use this page to view the attributes of a value set in a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Value Sets** > *value_set_name*.

This page shows the values of keyedReferences in the tModel that represents this value set. This page also shows the *Supported* status of the value set. All properties are read-only. To change the *Supported* status, use the Value sets collection page.

Unvalidatable:

Specifies whether this value set is unvalidatable. The value set tModel publisher sets this value, to indicate whether the value set is available for use by publish requests.

Checked:

Specifies whether this value set is checked. If you set this value to true, UDDI entities that reference this value set will be validated to ensure that their values are present in this value set.

Cached:

Specifies whether this value set is cached in this UDDI node.

Externally cacheable:

Specifies whether this value set is externally cacheable.

Externally validated:

Specifies whether this value set is externally validated.

Supported:

Specifies whether this value set is supported by policy in this UDDI node.

Last cached:

Specifies the date when this value set was last cached in the UDDI node.

Tier collection:

This page contains a list of the available tiers for the UDDI node. You can modify tiers, create new tiers and delete tiers from this page.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Tiers**.

This page shows the available tiers for the UDDI node. Clicking a tier name will show the General Properties for the specific tier as detailed in Tier settings. To delete a Tier from the list, select the relevant name and click *Delete*. Clicking *New* will take you to the General Properties page with the same properties as described in Tier settings.

One of the tiers in the collection will be marked as the default tier, indicated by *(default)* appearing next to the tier's name. The default tier will be assigned to UDDI publishers that are registered automatically when automatic user registration is turned on. To set the default tier, select the appropriate tier in the collection and press the *Set default* button. Note that it is not possible to delete a tier if it is currently marked as the default tier, or it is currently assigned to a UDDI publisher.

Name:

Specifies the name of the tier.

Description:

Specifies the descriptive text about the tier.

UDDI Tier settings:

This topic contains details of the general properties that you can configure for a UDDI publisher tier.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Tiers** > *tier_name*.

Name:

Specifies the name of the tier.

Required	Yes
Data type	String
Default	No default
Range	1 to 255

Description:

Specifies a description of the purpose or usage of this tier.

Data type	String
Default	No default
Range	0 to 255

Maximum properties:

For each of the maximum fields described below, the data is:

Required	Yes
Data type	Integer
Default	No default
Range	0 to 2147483647

Maximum businesses:

Specifies the maximum number of businesses that UDDI publishers in this tier are allowed to publish.

Maximum services per business:

Specifies the maximum number of services that UDDI publishers in this tier are allowed to publish for each business.

Maximum bindings per service:

Specifies the maximum number of bindings that UDDI publishers in this tier are allowed to publish for each service.

Maximum tModels:

Specifies the maximum number of tModels that UDDI publishers in this tier are allowed to publish.

Maximum publisher assertions:

Specifies the maximum number of publisher assertions that UDDI publishers in this tier are allowed to add.

UDDI Publisher collection:

This page shows the users that are currently registered as UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes >** *UDDI_node_id* **> [Additional Properties] UDDI Publishers**.

To create a UDDI publisher, click **New**. The "UDDI Publisher settings" on page 524 page, where you can enter details about the publisher, is displayed.

To register one or more existing WebSphere Application Server users as UDDI publishers, click **Create publishers**. The "Create UDDI Publishers" on page 523 page, where you can select users and modify their entitlements, is displayed.

You can assign multiple publishers to a tier without editing each publisher individually, by completing the following steps:

1. Select the appropriate publishers in the collection table.
2. From the tier list at the top of the collection table, select one of the tiers that is available on the UDDI node.
3. Click **Assign tier** to update the selected publishers.

To delete publishers, select them in the collection table and then click **Delete**.

After you register the users as UDDI publishers, you can edit their entitlements as described in "UDDI Publisher settings" on page 524.

User name:

Specifies the name of the UDDI publisher.

Tier:

Specifies the tier to which the UDDI publisher is assigned.

Create UDDI Publishers:

Use this page to register one or more existing WebSphere Application Server users as UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes >** *UDDI_node_id* **> [Additional Properties] UDDI Publishers** → **Create publishers**.

To register as UDDI publishers one or more users that are known to the application server, complete the following steps:

1. Enter a string to search for the required users. Use '*' to find all users.
2. Optionally, enter a number in the limit field to restrict the number of returned results.
3. Click **Search** to display a list of users that match the string.
4. Select the users that you want to register from the **Available** list and use the arrows to move them into the **Selected** list.
5. Use the entitlements listed under **General Properties** to give the UDDI publishers permission to perform specific actions. Set the entitlements by selecting the check box next to each entitlement.
6. Select a tier for the users from the **Tier** list.
7. Click **OK** to register the users as UDDI publishers with the specified entitlements and tier.

Note: If you are using a Lightweight Directory Access Protocol (LDAP) user registry, the format of the name that is given to each UDDI Publisher is defined by the **User ID map** value in the LDAP advanced settings. To view the LDAP advanced settings, click **Security** → **Secure administration, applications, and infrastructure**, under **User account repository** select **Standalone LDAP**

registry and click **Configure**, then under **Additional Properties** click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

After a user has been registered as a UDDI publisher, you can edit their entitlements by clicking the user name.

Allowed to publish keyGenerator with derived key:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a derived key.

The tModel:keyGenerator is a request for key space. An example of a legal derived key is uddi:tempuri.com:fish:buyingService where the key is based on the derivedKey ″uddi:tempuri.com:fish″. the string 'buyingService' is the key's key specific string (KSS).

Allowed to publish keyGenerator with domain keys:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a domain key.

Allowed to publish keyGenerator:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator.

If false, UDDI publishers cannot publish keyGenerators of any kind. In this situation all the entitlement settings are disregarded, regardless of how they are set.

Allowed to publish with UUID key:

Specifies whether the UDDI publisher has permission to publish elements with a UUID key.

Allowed to publish keyGenerator with UUID keys:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a UUID key.

Tier:

Specifies the tier to which the UDDI publisher is assigned.

UDDI Publisher settings:

Use this page to view and edit the properties of a UDDI publisher, or to create a new UDDI publisher.

You can view this administrative console page in two ways:

- If you want to view and edit the properties of an existing UDDI publisher click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **UDDI Publishers** > *user_name*
- If you want to create a new UDDI publisher click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **UDDI Publishers** → **New**

This page shows the entitlements and publication limits tier for a particular UDDI publisher.

User name:

Specifies the name of the UDDI publisher.

If you are creating a new UDDI publisher, enter the name of a user known to the application server. If you are viewing or editing the properties of an existing publisher, the user name cannot be changed.

Allowed to publish keyGenerator with derived key:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a derived key.

The tModel:keyGenerator is a request for key space. An example of a legal derived key is uddi:tempuri.com:fish:buyingService where the key is based on the derivedKey ″uddi:tempuri.com:fish″. the string 'buyingService' is the key's key specific string (KSS).

Data type	Boolean
Default	True (selected)

Allowed to publish keyGenerator with domain keys:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a domain key.

Data type	Boolean
Default	True (selected)

Allowed to publish keyGenerator:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator.

If you set this value to false, the UDDI publisher cannot publish keyGenerators of any kind. In this situation the following permissions will be disregarded irrespective of how they are set: **Allowed to publish keyGenerator with derived key**, **Allowed to publish keyGenerator with domain keys**, **Allowed to publish with UUID key** and **Allowed to publish keyGenerator with UUID keys**.

Data type	Boolean
Default	True (selected)

Allowed to publish with UUID key:

Specifies whether the UDDI publisher has permission to publish elements with a UUID key.

Data type	Boolean
Default	False (cleared)

Allowed to publish keyGenerator with UUID keys:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a UUID key.

Data type	Boolean
Default	False (cleared)

Tier:

Specifies the tier to which the UDDI publisher is assigned.

Policy groups:

This topic contains links to the detailed settings information for every policy group that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id*.

To the right of the page is a list of the Policy Groups that can be acted upon. Clicking on a specific group will open the page for the group required.

UDDI keying policy settings:

This topic contains details of the UDDI keying settings that you can configure for a UDDI registry.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Keying policies**.

Registry key generation:

Specifies whether publishers are allowed to publish key generator tModels. You can manage how publishers are allowed to publish key generator tModels by configuring the entitlements in the UDDI Publishers page.

Data type	Boolean
Default	True (selected)

Registry support of UUID keys:

Specifies whether publisher supplied uuidKeys are allowed in publish requests. You can manage how publishers are allowed to use uuidKeys by configuring the entitlements in the UDDI Publishers page.

Data type	Boolean
Default	False (cleared)

UDDI node API policy settings:

This topic contains details of the API settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **API policies**.

Note: This information applies only to UDDI Version 3. The settings for Versions 1 and 2 are not changeable; authentication tokens are required for publish requests, but not for inquiry requests (there are no custody transfer requests in Versions 1 or 2).

Authorization for inquiry:

Specifies whether authorization using the authInfo element is required for inquiry API requests. This setting is only relevant if the V3SOAP_Inquiry_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_Inquiry_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true an authorization token will be required to complete the request. If this setting is false, an authorization token is not required; if one is supplied it will be ignored and the request will be processed as if it was made by the default user defined in the UDDI node settings.

Typically, UDDI registries are configured not to require authorization for inquiry API requests.

Data type	Boolean
Default	False (cleared)

Authorization for publish:

Specifies whether authorization using the authInfo element is required for publish API requests. This setting is only relevant if the V3SOAP_Publish_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_Publish_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true an authorization token will be required to complete the request. If this setting is false, an authorization token is not required; if one is supplied it will be ignored and the request will be processed as if it was made by the default user defined in the UDDI node settings.

Typically, UDDI registries are configured to require authorization for publish API requests.

Data type	Boolean
Default	True (selected)

Authorization for custody transfer:

Specifies whether authorization using the authInfo element is required for custody transfer API requests. This setting is only relevant if the V3SOAP_CustodyTransfer_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_CustodyTransfer_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true an authorization token will be required to complete the request. If this setting is false, an authorization token is not required; if one is supplied it will be ignored and the request will be processed as if it was made by the default user defined in the UDDI node settings.

Typically, UDDI registries are configured to require authorization for custody transfer API requests.

Data type	Boolean
Default	True (selected)

UDDI user policy settings:

This topic contains details of the user policy settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **User policies**.

Allow transfer of ownership:

When true, data ownership can be transferred between owners within the UDDI node.

Data type	Boolean
Default	True (selected)

UDDI data custody policy settings:

This topic contains details of the data custody settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Data custody policies**.

Transfer token expiration period:

Specifies the length of time from the issue of a transfer token, in minutes, after which that token expires.

If you set this value too high, you might expose the UDDI registry to a risk of misuse.

Data type	Integer
Default	1440
Range	1 to 2147483647 (for all intents and purposes, unlimited)

UDDI value set policy:

This topic contains details of the value set policy settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* >**Value set policies**.

Enable checked value sets:

Specifies whether checked value sets are supported. If you set this value to false, publish requests of value set tModels that contain a 'checked' keyedReference will be rejected.

Data type	Boolean
Default	True (selected)

UDDI node miscellaneous:

This topic contains details of the miscellaneous settings that you can configure for a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Miscellaneous policies**.

Node generates discoveryURLs:

Specifies whether a UDDI node can establish a policy on whether it generates discoveryURLs.

Data type	Boolean
Default	False (cleared)

Node supports HTTP Get Service:

Specifies whether the UDDI node supports an HTTP GET service for access to the XML representations of UDDI data structures.

Data type	Boolean
Default	True (selected)

URL prefix for V3 GET servlet:

Specifies the prefix for the URL to the version 3 GET servlet that is used to retrieve the XML representation of a published entity. This property applies to UDDI version 3 API requests only.

The format of the prefix is http://hostname:port/uddiv3soap/, where uddisoap is the context root of the UDDI version 3 SOAP servlet.

When a businessEntity is published, if you have set **Node generates discovery URLs** to true, the discoveryURL value is generated based on this prefix value. Otherwise, the discoveryURL value will be empty.

The UDDI Version 3 specification recommends that you do **not** enable generation of discoveryURLs as they can affect the use of digital signatures. If you do enable generation of discoveryURLs, it is recommended that you do not change the URL prefix afterwards, otherwise discoveryURLs generated using the earlier URL prefix will no longer work.

Data type	URL
Default	http://localhost:9080/uddiv3soap/

Overview of the UDDI registry Administrative Interface

The UDDI registry Administrative Interface allows you to inspect and manage the runtime configuration of a UDDI application. This includes managing the information and the activation state about a UDDI node, updating properties and policies, setting publish tier limits, registration of UDDI publishers, and controlling value set support. The operations of the UDDI registry Administrative Interface can be read and invoked using standard JMX (Java Management Extensions) interfaces.

The use of JMX is explained in Using administrative programs (JMX). See the UDDI registry Administrative Interface for full details on using the UDDI administrative interface.

Backing up and restoring the UDDI registry database

If you want to protect the data in your UDDI registry database, you can back up and restore the database using the facilities of the database product your UDDI node is on.

Cloudscape

To backup a Cloudscape UDDI registry database, first ensure that the UDDI application is stopped (and hence, not accessing the Cloudscape database), and ensure that no other application is using the Cloudscape UDDI30 database. Make a copy of the UDDI30 directory using the file system that the directory resides upon.

To restore the database, replace the UDDI30 file structure with the back up. Note that any updates made since the back up was taken will be lost.

Non-Cloudscape

Use the appropriate backup and restore tools for the database being used to contain the UDDI registry. To use these tools, refer to the documentation for the database product.

Chapter 13. Data access resources

Task overview: Accessing data from applications

Various enterprise information systems (EIS) use different methods for storing data. These *backend* data stores might be relational databases, procedural transaction programs, or object-oriented databases. IBM WebSphere Application Server provides several options for accessing an information system's backend data store:

- Programming directly to the database through the JDBC 2.0 optional package API or the JDBC 3.0 API.
- Programming to the procedural backend transaction through various J2EE Connector Architecture (JCA) 1.0 or 1.5 compliant connectors.
- Programming in the bean-managed persistence (BMP) bean or servlets indirectly accessing the backend store through either the JDBC API or JCA compliant connectors.
- Using container-managed persistence (CMP) beans.
- Using embedded Structured Query Language in Java (SQLJ) support with applications that use DB2 as a backend database.
- Using the IBM data access beans, which also use the JDBC API, but give you a rich set of features and function that hide much of the complexity associated with accessing relational databases.

For all of these options, *except for using the JCA 1.0 or 1.5 compliant connectors*, the prerequisite Web site details which databases and drivers are currently supported. Consult the IBM Web address: http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html .

1. Develop data access applications. Develop your application to access data using the various ways available through the WebSphere Application Server. You can access data through APIs, container-managed persistence beans, bean-managed persistence beans, session beans, or Web components.
2. Assemble data access applications using the assembly tool. Assemble your application by creating and mapping resource references.
3. Prepare for deployment: Ensure that the appropriate database objects are available. Create or configure any databases or tables required, set necessary configuration parameters to handle expected load, and configure any necessary JDBC providers and data source objects for servlets, enterprise beans, and client applications to use.
4. Install the application on your application server.

Resource adapter

A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS).

A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

An application server vendor extends its system once to support the J2EE Connector Architecture (JCA) and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard resource adapter with the capability to plug into any application server that supports the connector architecture.

WebSphere Application Server provides the WebSphere Relational Resource Adapter implementation. This resource adapter provides data access through JDBC calls to access the database dynamically. The connection management is based on the JCA connection management architecture. It provides connection pooling, transaction, and security support. WebSphere Application Server supports JCA versions 1.0 and 1.5.

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

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

J2EE Connector Architecture resource adapters

A J2EE Connector Architecture (JCA) resource adapter is any resource adapter conforming to the JCA Specification.

The product supports any resource adapter that implements version 1.0 or 1.5 of this specification. IBM supplies resource adapters for many enterprise systems separately from the WebSphere Application Server package, including (but not limited to): the Customer Information Control System (CICS), Host On-Demand (HOD), Information Management System (IMS), and Systems, Applications, and Products (SAP) R/3 .

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

WebSphere relational resource adapter settings

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

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

To view this administrative console page, click **Resources** > **Resource adapters** > **Resource adapters** > **WebSphere Relational Resource Adapter**.

Name:

Specifies the name of the resource provider.

Data type String

Description:

Specifies a description of the relational resource adapter.

Data type String

Archive path:

Specifies the path to the Resource Adapter Archive (RAR) file containing the module for this resource adapter.

Data type String

Class path:

Specifies a list of paths or Java Archive (JAR) file names, which together form the location for the resource provider classes.

Data type String

Native path:

Specifies a list of paths that forms the location for the resource provider native libraries.

Data type String

WebSphere Relational Resource Adapter

The WebSphere Relational Resource Adapter (RRA) provides enterprise applications deployed on WebSphere Application Server access to relational databases.

The WebSphere RRA is installed and runs as part of WebSphere Application Server, and needs no further administration.

The RRA supports both the configuration and use of JDBC data sources and J2EE Connection Architecture (JCA) connection factories. The RRA supports the configuration and use of data sources implemented as either JDBC data sources or J2EE Connector Architecture connection factories. Data sources can be used directly by applications, or they can be configured for use by container-managed persistence (CMP) entity beans.

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

- For information about resource adapters, see “Resource adapter” on page 557
- For information about resource adapters and data access, see “Data access portability features”
- For RRA settings, see “WebSphere relational resource adapter settings” on page 558
- For information about CMP connection factories, see “Connection factory” on page 563
- For information about enterprise beans, see EJB applications

Data access portability features

The WebSphere Application Server relational resource adapter (RRA) provides a portability feature that enables applications to access data from different databases without changing the application. In addition, WebSphere Application Server enables you to plug in a data source that is not supported by WebSphere persistence. However, the data source *must* be implemented as either the *XADataSource* type or the *ConnectionPoolDataSource* type, and it must be in compliance with the JDBC 2.x specification.

You can achieve application portability through the following:

DataStoreHelper interface

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

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

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

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

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

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

WSCallHelper class

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

- **jdbcCall() method**

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

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

- **jdbcPass() method**

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

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

  ```
  oracle.sql.ArrayDescriptor ArrayDescriptor.createDescriptor(java.lang.String, java.sql.Connection)
  oracle.sql.ARRAY new ARRAY(oracle.sql.ArrayDescriptor, java.sql.Connection, java.lang.Object)
  oracle.xml.sql.query.OracleXMLQuery(java.sql.Connection, java.lang.String)
  oracle.sql.BLOB.createTemporary(java.sql.Connection, boolean, int)
  oracle.sql.CLOB.createTemporary(java.sql.Connection, boolean, int)
  oracle.xdb.XMLType.createXML(java.sql.Connection, java.lang.String)
  ```

  The following code examples demonstrate the difference between a call to the XMLType.createXML() method over a direct connection to Oracle, and a call to the same method within WebSphere Application Server.

  1. Over a direct connection:

     ```
     XMLType poXML = XMLType.createXML(conn, poString);
     ```

  2. Within Application Server, using the jdbcPass() method:

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

There are two different jdbcPass() methods available, one for use in invoking static methods, another for use when invoking non-static methods. See the API documentation **WSCallHelper** topic.

Note: Because of the possible problems that can occur by passing an underlying object to a method, WebSphere Application Server strictly controls which methods are allowed to be invoked using the jdbcPass() method support. If you require support for a method that is not listed previously in this document, please contact WebSphere Application Server support with information on the method you require.

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

Example: Developing your own DataStoreHelper class: The DataStoreHelper interface supports each data store platform plugging in its own private data store specific functions that are used by the Relational Resource Adapter run time.

```
package com.ibm.websphere.examples.adapter;

import java.sql.SQLException;
import javax.resource.ResourceException;

import com.ibm.websphere.appprofile.accessintent.AccessIntent;
import com.ibm.websphere.ce.cm.*;
import com.ibm.websphere.rsadapter.WSInteractionSpec;

/**
* Example DataStoreHelper class, demonstrating how to create a user-defined DataStoreHelper.
* Implementation for each method is provided only as an example.  More detail would likely be
* required for any custom DataStoreHelper created for use by a real application.
*/
public class ExampleDataStoreHelper extends com.ibm.websphere.rsadapter.GenericDataStoreHelper
{
    static final long serialVersionUID = 8788931090149908285L;


    public ExampleDataStoreHelper(java.util.Properties props)
    {
        super(props);

        // Update the DataStoreHelperMetaData values for this helper.
        getMetaData().setGetTypeMapSupport(false);

        // Update the exception mappings for this helper.
        java.util.Map xMap = new java.util.HashMap();

        // Add an Error Code mapping to StaleConnectionException.
        xMap.put(new Integer(2310),  StaleConnectionException.class);
        // Add an Error Code mapping to DuplicateKeyException.
        xMap.put(new Integer(1062),  DuplicateKeyException.class);
        // Add a SQL State mapping to the user-defined ColumnNotFoundException
        xMap.put("S0022",            ColumnNotFoundException.class);
        // Undo an inherited StaleConnection SQL State mapping.
        xMap.put("S1000",            Void.class);

        setUserDefinedMap(xMap);

        // If you are extending a helper class, it is
        // normally not necessary to issue 'getMetaData().setHelperType(...)'
        // because your custom helper will inherit the helper type from its
```

```
        //  parent class.


         }

    public void doStatementCleanup(java.sql.PreparedStatement stmt) throws SQLException
    {
        // Clean up the statement so it may be cached and reused.

        stmt.setCursorName("");
        stmt.setEscapeProcessing(true);
        stmt.setFetchDirection(java.sql.ResultSet.FETCH_FORWARD);
        stmt.setMaxFieldSize(0);
        stmt.setMaxRows(0);
        stmt.setQueryTimeout(0);
    }


    public int getIsolationLevel(AccessIntent intent) throws ResourceException
    {
        // Determine an isolation level based on the AccessIntent.

        if (intent == null) return java.sql.Connection.TRANSACTION_SERIALIZABLE;

        return intent.getConcurrencyControl() == AccessIntent.CONCURRENCY_CONTROL_OPTIMISTIC ?
               java.sql.Connection.TRANSACTION_READ_COMMITTED :
               java.sql.Connection.TRANSACTION_REPEATABLE_READ;
    }
        public int getLockType(AccessIntent intent) {
           if ( intent.getConcurrencyControl() == AccessIntent.CONCURRENCY_CONTROL_PESSIMISTIC) {
              if ( intent.getAccessType() == AccessIntent.ACCESS_TYPE_READ ) {
                  return WSInteractionSpec.LOCKTYPE_SELECT;
              }
              else {
                  return WSInteractionSpec.LOCKTYPE_SELECT_FOR_UPDATE;
               }
           }
           return WSInteractionSpec.LOCKTYPE_SELECT;
        }


    public int getResultSetConcurrency(AccessIntent intent) throws ResourceException
    {
        // Determine a ResultSet concurrency based on the AccessIntent.

        return intent == null || intent.getAccessType() == AccessIntent.ACCESS_TYPE_READ ?
               java.sql.ResultSet.CONCUR_READ_ONLY :
               java.sql.ResultSet.CONCUR_UPDATABLE;
    }


    public int getResultSetType(AccessIntent intent) throws ResourceException
    {
        // Determine a ResultSet type based on the AccessIntent.

        if (intent == null) return java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE;

        return intent.getCollectionAccess() == AccessIntent.COLLECTION_ACCESS_SERIAL ?
               java.sql.ResultSet.TYPE_FORWARD_ONLY :
               java.sql.ResultSet.TYPE_SCROLL_SENSITIVE;
    }
}
```

ColumnNotFoundException

```
package com.ibm.websphere.examples.adapter;

import java.sql.SQLException;
import com.ibm.websphere.ce.cm.PortableSQLException;

/**
* Example PortableSQLException subclass, which demonstrates how to create a user-defined
* exception for exception mapping.
*/
public class ColumnNotFoundException extends PortableSQLException
{
    public ColumnNotFoundException(SQLException sqlX)
    {
        super(sqlX);
    }
}
```

Connection factory

An application component uses a *connection factory* to access a connection instance, which the component then uses to connect to the underlying enterprise information system (EIS).

Examples of connections include database connections, Java Message Service connections, and SAP R/3 connections.

CMP connection factories collection

Use this page to view existing CMP connection factories settings.

These connection factories are used by a container-managed persistence (CMP) bean to access any backend data store. A CMP connection factory is used by EJB model 2.x Entities with CMP version 2.x. Connection factories listed on this page are created automatically under the WebSphere Relational Resource Adapter when you check the box *Use this Data Source in container managed persistence (CMP)* in the General Properties area on the Data Source page. You cannot modify the settings for a CMP connection factory, and you cannot delete CMP connection factories from this collection. To remove the CMP connection factory object, you must navigate to the data source associated with the CMP connection factory and uncheck the *Use this Data Source for CMP* check box.

To view this administrative console page, click **Resources** > **Resource Adapters** > **Resource Adapters** >*WebSphere Relational Resource Adapter* > **CMP connection factories**.

Name:

Specifies a list of the display names for the resources.

Data type String

JNDI Name:

Specifies the JNDI name of the resource.

Data type String

Description:

Specifies a description for the resource.

Data type String

Category:

Specifies a category string which can be used to classify or group the resource.

Data type String

CMP connection factory settings:

Use this page to view the settings of a connection factory that is used by a CMP bean to access any backend data store. This connection factory is only in ″read″ mode. It cannot be modified or deleted.

To view this administrative console page, click **Resources** >**Resource Adapters** > **Resource Adapters** > *WebSphere Relational Resource Adapter*> **CMP Connection Factories** > *connection_factory*

Name:

Specifies the display name for the resource.

Data type String

JNDI name:

Specifies the JNDI name of the resource.

Data type String

Description:

Specifies a description for the resource.

Data type String

Category:

Specifies a category string which can be used to classify or group the resource.

Data type String

Authentication Preference:

Specifies which of the authentication mechanisms that are defined for the corresponding resource adapter applies to this connection factory. This property is deprecated starting with version 6.0.

For example, if two authentication mechanism entries are defined for a resource adapter (*KerbV5* and *Basic Password*), this specifies one of those two types. If the authentication mechanism preference specified is not an authentication mechanism available on the corresponding resource adapter, it is ignored.

Data type String

Component-managed authentication alias:

References authentication data for component-managed signon to the resource.

Data type Drop-down list

Container-managed authentication alias:

References authentication data for container-managed signon to the resource. This property is deprecated starting with version 6.0.

Data type Drop-down list

JDBC providers

Installed applications use JDBC providers to interact with relational databases.

The JDBC provider object supplies the specific JDBC driver implementation class for access to a specific vendor database. To create a pool of connections to that database, you associate a data source with the JDBC provider. Together, the JDBC provider and the data source objects are functionally equivalent to the J2EE Connector Architecture (JCA) connection factory, which provides connectivity with a non-relational database.

For a current list of supported providers, see the WebSphere Application Server prerequisite Web site.

See also Hardware and software requirements for more information. For detailed descriptions of the providers, including the supported data source classes and their required properties, refer to "Vendor-specific data sources minimum required settings" on page 687.

Data sources

Installed applications use a *data source* to obtain connections to a relational database. A data source is analogous to the J2EE Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

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

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

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

In WebSphere Application Server releases prior to version 5.0, the function of data access was provided by a single connection manager (CM) architecture. This connection manager architecture remains available to support J2EE 1.2 applications, but another connection manager architecture is provided, based on the JCA architecture supporting the new J2EE 1.3 application style (also for J2EE 1.4 applications).

These two separate architectures are represented by two types of data sources. To choose the right data source, administrators must understand the nature of their applications, EJB modules, and enterprise beans.

- Data source (WebSphere Application Server V4) - This data source runs under the original CM architecture. Applications using this data source behave as if they were running in Version 4.0.
- Data source - This data source uses the JCA standard architecture to provide support for J2EE version 1.3 and 1.4 applications. It runs under the JCA connection manager and the relational resource adapter.

Choice of data source

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

Data access beans

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

They are Java classes written to the Enterprise JavaBeans specification.

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

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

Feature
: **Details**

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

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

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

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

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

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

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

Connection management architecture

The connection management architecture for both relational and procedural access to enterprise information systems (EIS) is based on the J2EE Connector Architecture (JCA) specification. The Connection Manager (CM), which pools and manages connections within an application server, is capable of managing connections obtained through both resource adapters (RAs) defined by the JCA specification, and data sources defined by the Java Database Connectivity (JDBC) 2.0 (and later) Extensions specification.

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

Applications migrating from previous versions of WebSphere Application Server might experience some behavioral differences because of the specification changes from various J2EE requirements levels. These differences are not related to the adoption of the JCA architecture.

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

Connection pooling

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

WebSphere Application Server supports JDBC 3.0 APIs for connection pooling and connection reuse. The connection pool is used to direct JDBC calls within the application, as well as for enterprise beans using the database.

Benefits of connection pooling

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

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

When to use connection pooling

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

How connections are pooled together

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

A separate instance of a given configured connection pool is created on each application server that uses that data source or connection factory. For example, if you run a three server cluster in which all of the servers use *myDataSource*, and *myDataSource* has a maximum connections setting of 10, then you can generate up to 30 connections (three servers times 10 connections). Be sure to consider this fact when determining how many connections to your backend resource you can support.

Other considerations for determining the maximum connections setting:
- Each entity bean transaction requires an additional database connection, dedicated to handling the transaction.
- On UNIX platforms, a separate DB2 process is created for each connection; these processes quickly affect performance on systems with low memory and cause errors.
- If clones are used, one data pool exists for each clone.

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

Avoiding a deadlock

Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:
- Each thread has its first database connection, and all are in use.
- Each thread is waiting for a second database connection, and none would become available since all threads are blocked.

To prevent the deadlock in this case, the maximum connections value for the database connection pool should be increased by at least one. Doing this allows for at least one of the waiting threads to obtain its second database connection and to avoid a deadlock.

To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require *C* concurrent database connections per thread, the connection pool must support at least the following number of connections, where *T* is the maximum number of threads.

```
T * (C - 1) + 1
```

The connection pool settings are directly related to the number of connections that the database server is configured to support.

If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed in the `stderr.log` file.

Deferred Enlistment: In the WebSphere Application Server environment, *deferred enlistment* is a term used to refer to the technique of waiting until a connection is first used to enlist it in its unit of work (UOW) scope.

In one example, the technique works like this: a component calls getConnection() from within a global transaction, and at some point later in time, the component uses the connection. The call that uses the connection is intercepted, and the XA resource for that connection is enlisted with the transaction service (which in turn calls XAResource.start()). Next, the actual call is sent to the resource manager.

In contrast, if a component gets a connection within a global transaction without deferred enlistment, then the connection is enlisted in the transaction and has all the overhead associated with that transaction. For XA connections, this includes the two phase commit (2PC) protocol to the resource manager. Deferred enlistment offers better performance in the case where a connection is obtained, but not used within the UOW scope. This saves all the overhead of participating in the UOW when it is not needed.

The WebSphere Application Server relational resource adapter automatically supports deferred enlistment without any additional configuration needed.

Lazy Transaction Enlistment Optimization: The J2EE Connector Architecture (JCA) Version 1.5 specification calls the deferred enlistment technique *lazy transaction enlistment optimization*. This support comes through a marker interface (LazyEnlistableManagedConnection) and a new method on the connection manager (LazyEnlistableConnectionManager()):

```
package javax.resource.spi;
import javax.resource.ResourceException;
import javax.transaction.xa.Xid;

interface LazyEnlistableConnectionManager { // application server
    void lazyEnlist(ManagedConnection) throws ResourceException;
}

interface LazyEnlistableManagedConnection { // resource adapter
}
```

A resource adapter is not required to support this functionality. Check with the resource adapter provider if you need to know if the resource adapter provides this functionality.

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

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

- UseTime

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

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

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

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

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

In WebSphere Application Server Version 5.x, the JCAStats interface was implemented by the J2CResourceAdapter Mbean, and the JDBCStats interface was implemented by the JDBCProvider Mbean. The JCAConnectionStats and JDBCConnectionStats interfaces are not implemented because they collect statistics for nonpooled connections, which are not present in the JCA 1.0 Specification. JCAConnectionPoolStats, and JDBCConnectionPoolStats do not have a direct implementing Mbean; those statistics are gathered through a call to PMI. A J2C resource adapter, and JDBC provider each contain a list of ConnectionFactory or DataSource ObjectNames, respectively. The ObjectNames are used by PMI to find the appropriate connection pool in the list of PMI modules.

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

Connection life cycle

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

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

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

- the application has called the data source or connection factory getConnection() method (the *getConnection* condition)
- a free connection is available in the pool with matching properties (the *freeConnectionAvailable* condition)
- and one of the two following conditions are true:
  - the getConnection() request is on behalf of a resource reference that is marked unsharable
  - the getConnection() request is on behalf of a resource reference that is marked shareable but no shareable connection in use has the same properties.

This transition description follows:

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

Here is a list of guarding conditions and descriptions.

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

Condition	Description
unusedTimeoutExpired	Connection is in the free pool and not in use past its unused timeout value.

Getting connections

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

DoesNotExist

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

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

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

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

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

InFreePool

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

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

This transition states that a connection is placed in use from the free pool if:
- the application has issued a getConnection() call
- a connection is available for use in the connection pool (freeConnectionAvailable),
- and one of the following is true:
  - the request is for an unsharable connection (unsharableConnectionRequest)

– no connection with the same sharing properties is already in use in the transaction. (NOT(sharableConnectionAvailable)).

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

InUse

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

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

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

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

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

Returning connections

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

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

Conditions that cause the transition from the InUse state are:

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

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

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

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

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

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

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

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

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

Unshareable and shareable connections

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

You can share a shareable connection with other components within the same transaction as long as each *getConnection()* request has the same connection properties. To enable connection sharing for data sources, the following connection properties must be the same:

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

To enable connection sharing for resource adapters within the same transaction, the following connection properties must be the same:
- JNDI name. While not actually a connection property, this requirement simply means that you can only share connections from the same resource adapter in the same server.
- Resource authentication

In addition, the *ConnectionSpec* object used to get the connection must also be the same. For more information on sharing a connection with a CMP bean, see Sharing a connection with a CMP bean.

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

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

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

For WebSphere Application Server, all sharing of connections is relative to the current Unit of Work (UOW) boundary. Anyone within a specific transaction, when getting a connection from a specific connection pool, gets a handle to the same physical connection (if the sharing properties are the same).

Sharing a connection with a CMP bean

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

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

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

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

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

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

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

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

  – Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level that is specified on the resource reference to look up a data source and a connection. This option is more of a manual process, and the isolation level might be different from database to database. For more information refer to the isolation level and access intent mapping table: Access intent isolation levels and update locks and the Isolation level and resource reference section.

- **Sharing a connection between CMP and a JDBC application that is used by a servlet or a session bean**

  Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level specified on the resource reference to look up a data source and a connection. For more information refer to Access intent isolation levels and update locks and Isolation level and resource reference.

Factors that determine sharing

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

- Only connections acquired with the same resource reference (resource-ref) that specifies the res-sharing-scope as shareable are candidates for sharing. The resource reference properties of res-sharing-scope and res-auth and the IBM extension isolationLevel help determine if it is possible to share a connection. IBM extension isolationLevel is stored in IBM deployment descriptor extension file; for example: ibm-ejb-jar-ext.xmi.
- You can only share connections that are requested with the same properties.
- Connection Sharing only occurs between different component instances if they are within a transaction (container- or user-initiated transaction).
- Connection Sharing only occurs within a sharing boundary. Current sharing boundaries include *Transactions* and *LocalTransactionContainment* (LTC) boundaries.
- Connection Sharing rules within an LTC Scope:
  – For shareable connections, only *Connection Reuse* is allowed within a single component instance. Connection reuse occurs when the following actions are taken with a connection: get, use, commit/rollback, close; get, use, commit/rollback, close. Note that if you use the LTC resolution-control of *ContainerAtBoundary* then no start/commit is needed because that action is handled by the container.

    The connection returned on the second *get* is the same connection as that returned on the first *get* (if the same properties are used). Because the connection use is serial, only one connection handle to the underlying physical connection is used at a time, so true connection sharing does not take place. The term *"reuse"* is more accurate.

    More importantly, the *LocalTransactionContainment* boundary enclosing both *get* actions is not complete; no cleanUp() method is invoked on the ManagedConnection object. Therefore the second *get* action inherits all of the connection properties set during the first getConnection() call.
- Shareable connections between transactions (either container-managed transactions (CMT), bean-managed transactions (BMT), or LTC transactions) follow these caching rules:

  - In general, setting properties on shareable connections is not allowed because a user of one connection handle might not anticipate a change made by another connection handle. This limitation is part of the J2EE 1.3 standard.
  - General users of resource adapters can set the connection properties on the connection factory getConnection() call by passing them in a *ConnectionSpec*.

    However, the properties set on the connection during one transaction are not guaranteed to be the same when used in the next transaction. Because it is not valid to share connections outside of a sharing scope, connection handles are moved off of the physical connection with which they are currently associated when a transaction ends. That physical connection is returned to the free connection pool. Connections are cleaned before going in the free pool. The next time the handle is used, it is automatically associated with an appropriate connection. The appropriateness is based on the security login information, connection properties, and (for the JDBC API) the *isolation level* specified in the extended resource reference, passed in on the original request that returned the current handle. Any properties set on the connection after it was retrieved are lost.
  - For JDBC users, WebSphere Application Server provides an extension to enable passing the connection properties through the ConnectionSpec.

    Use caution when setting properties and sharing connections in a local transaction scope. Ensure that other components with which the connection is shared are expecting the behavior resulting from your settings.
- You cannot set the isolation level on a shareable connection for the JDBC API using a relational resource adapter in a global transaction. The product provides an extension to the resource reference to enable you to specify the isolation level. If your application requires the use of multiple isolation levels, create multiple resource references and map them to the same data source or connection factory.

Connection sharing violations

There is a new exception, the `SharingViolation` exception, that the resource adapter can issue whenever an operation violates sharing requirements. Possible violations include changing connection attributes, security settings, or isolation levels, among others. When such a mutable operation is performed against a managed connection, the `SharingViolation` exception can occur when both of the following conditions are true:

- The number of connection handles associated with the managed connection is more than one.
- The managed connection is associated with a transaction, either local or XA.

Both the component and the J2C run time might need to detect this `SharingViolation` exception, depending on when and how the managed connection becomes unshareable. If the managed connection becomes unshareable because of an operation through the connection handle (for example, you change the isolation level), then the component needs to process the exception. If the managed connection becomes unshareable without being recognized by the application server (due to some component interaction with the connection handle), then the resource adapter can reject the creation of a connection handle by issuing the `SharingViolation` exception.

Connection handles

A connection handle is a representation of a physical connection.

To use a backend resource (such as a relational database) in WebSphere Application Server you must get a connection to that resource. When you call the *getConnection()* method, you get a *connection handle* returned. The handle is not the physical connection. The physical connection is managed by the connection manager.

There are two significant configurations that affect how connection handles are used and how they behave. The first is the *res-sharing-scope*, which is defined by the resource-reference used to look up the DataSource or Connection Factory. This property tells the connection manager whether or not you can share this connection.

The second factor that affects connection handle behavior is the *usage pattern*. There are essentially two usage patterns. The first is called the *get/use/close* pattern. It is used within a single method and without calling another method that might get a connection from the same data source or connection factory. An application using this pattern does the following:
1. gets a connection
2. does its work
3. commits (if appropriate)
4. closes the connection.

The second usage pattern is called the *cached handle* pattern. This is where an application:
1. gets a connection
2. begins a global transaction
3. does work on the connection
4. commits a global transaction
5. does work on the connection again

A cached handle is a connection handle that is held across transaction and method boundaries by an application. Keep in mind the following considerations for using cached handles:

- Cached handle support requires some additional connection handle management across these boundaries, which can impact performance. For example, in a JDBC application, *Statements*, *PreparedStatements*, and *ResultSets* are closed implicitly after a transaction ends, but the connection remains valid.
- You are encouraged **not** to cache the connection across the transaction boundary for shareable connections; the get/use/close pattern is preferred.
- Caching of connection handles across servlet methods is limited to JDBC and Java Message Service (JMS) resources. Other non-relational resources, such as Customer Information Control System (CICS) or IMS objects, currently cannot have their connection handles cached in a servlet; you need to get, use, and close the connection handle within each method invocation. (This limitation only applies to single-threaded servlets because multithreaded servlets do not allow caching of connection handles.)
- You **cannot** pass a cached connection handle from one instance of a data access client to another client instance. Transferring between client instances creates the problematic contingency of one instance using a connection handle that is referenced by another. This relationship can only cause problems because connection handle management code processes tasks for each client instance *separately*. Hence, connection handle transfers result in run-time scenarios that trigger exceptions. For example:
  1. The application code of a client instance that receives a transferred handle closes the handle.
  2. If the client instance that retains the original reference to the handle tries to reclaim it, the application server issues an exception.

The following code segment shows the cached connection pattern.

```
Connection conn = ds.getConnection();
ut.begin();
conn.prepareStatement("....."); --> Connection runs in global transaction mode
...
ut.commit();
conn.prepareStatement(".....");   ---> Connection still valid but runs in autoCommit(True);
...
```

Unshareable connections

Some characteristics of connection handles retrieved with a *res-sharing-scope* of **unshareable** are described in the following sections.

- **The possible benefits of unshared connections**
  - Your application always maintains a direct link with a physical connection (managed connection).

  - The connection always has a one-to-one relationship between the connection handle and the managed connection.
  - In most cases, the connection does not close until the application closes it.
  - You can use a cached unshared connection handle across multiple transactions.
  - The connection can have a performance advantage in some cached handle situations. Because unshared connections do not have the overhead of moving connection handles off managed connections at the end of the transaction, there is less overhead in using a cached unshared connection.
- **The possible drawbacks of unshared connections**
  - Inefficient use of your connection resources. For example, if within a single transaction you get more than one connection (with the same properties) using the same data source or connection factory (same resource-ref) then you use multiple physical connections when you use unshareable connections.
  - Wasted connections. It is important not to keep the connection handle open (that is, your application does not call the *close()* method) any longer then it is needed. As long as an unshareable connection is open, the physical connection is unavailable to any other component, even if your application is not currently using that connection. Unlike a shareable connection, an ushareable connection is not closed at the end of a transaction or servlet call.
  - Deadlock considerations. Depending on how your components interact with the database within a transaction, using unshared connections can lead to deadlock in the database. For example, within a transaction, component A gets a connection to data source X and updates table 1, and then calls component B. Component B gets another connection to data source X, and updates/reads table 1 (or even worse the same row as component A). In some circumstances, depending on the particular database, its locking scheme, and the transaction isolation level, a deadlock can occur.

    In the same scenario, but with a *shared* connection, deadlock does not occur because all the work is done on the same connection. It is worth noting that when writing code that uses shared connections, you use a strategy that calls for multiple work items to be performed on the same connection, possibly within the same transaction. If you decide to use an unshareable connection, you must set the *maximum connections* property on the connection factory or data source correctly. An exception might occur for waiting connection requests if you exceed the maximum connections value, and unshareable connections are not being closed before the connection wait time-out is exceeded.

Shareable connections

Some characteristics of connection handles that are retrieved with a *res-sharing-scope* of **shareable** are described in the following sections.

- **The possible benefits of shared connections**
  - Within an instance of connection sharing, application components can share a managed connection with one or more connection handles, depending on how the handle is retrieved and which connection properties are used.
  - They can more efficiently use resources. Shareable connections are not valid outside of their sharing boundary. For this reason, at the end of a sharing boundary (such as a transaction) the connection handle is no longer associated with the managed connection it was using within the sharing boundary (this applies only when using the cached handle pattern). The managed connection is returned to the free connection pool for reuse. Connection resources are not held longer than the end of the current sharing scope.

    If the cached handle pattern is used, then the next time the handle is used within a new sharing scope, the application server run time ensures that the handle is reassociated with a managed connection that is appropriate for the current sharing scope, and has the same properties with which the handle was originally retrieved. Remember that it is not appropriate to change properties on a shareable connection. If properties are changed, other components that share the same connection might experience unexpected behavior. Futhermore, when using cached handles, the value of the changed property might not be remembered across sharing scopes.
- **The possible drawbacks of shared connections**

– Sharing within a single component (such as an enterprise bean and its related Java objects) is not always supported. The current specification allows resource adapters the choice of only allowing one active connection handle at a time.

  If a resource adapter chooses to implement this option then the following scenario results in an *invalid handle exception*: A component using shareable connections gets a connection and uses it. Without closing the connection, the component calls a utility class (Java object) that gets a connection handle to the same managed connection and uses it. Because the resource adapter only supports one active handle, the first connection handle is no longer valid. If the utility object returns without closing its handle, the first handle is not valid and triggers an exception at any attempt to use it.

  Note: This exception occurs only when calling a utility object (a Java object).

  Not all resource adapters have this limitation; it occurs only in certain implementations. The WebSphere Relational Resource Adapter (RRA) does not have this limitation. Any data source used through the RRA does not have this limitation. If you encounter a resource adapter with this limitation you can work around it by serializing your access to the managed connection. If you always close your connection handle before getting another (or close your handle before calling code that gets another handle), and before returning from a method, you can allow two pieces of code to share the same managed connection. You simply cannot use the connection for both events at the same time.
– Trying to change the *isolation level* on a shareable JDBC-based connection in a global transaction (that is supported by the RRA) causes an exception. The correct way to get connections with different transaction isolation levels is by configuring the IBM extended resource-reference.
– Closing connection handles for shareable connections by an application is NOT supported and causes errors. However, you can avoid this limitation by using the Relational Resource Adapter.

Lazy connection association optimization

In WebSphere Application Server Version 5.0, the Java 2 Platform, Enterprise Edition (J2EE) Connector (J2C) connection manager implemented *smart handle* support. This technology enables allocation of a connection handle to an application while the managed connection associated with that connection handle is used by other applications (assuming that the connection is not being used by the original application). This concept is part of the J2EE Connector Architecture (JCA) 1.5 specification. (You can find it in the JCA 1.5 specification document in the section entitled ″Lazy Connection Association Optimization.″) Smart handle support introduces use a method on the ConnectionManager object, the *LazyAssociatableConnectionManager()* method, and a new marker interface, the *DissociatableManagedConnection* class. You must configure the provider of the resource adapter to make this functionality available in your environment. (In the case of the RRA, WebSphere Application Server itself is the provider.) The following code snippet shows how to include smart handle support:

```
package javax.resource.spi;
import javax.resource.ResourceException;

interface LazyAssociatableConnectionManager { // application server
    void associateConnection(
        Object connection, ManagedConnectionFactory mcf,
        ConnectionRequestInfo info) throws ResourceException;
}

interface DissociatableManagedConnection { // resource adapter
    void dissociateConnections() throws ResourceException;
}
```

This DissociatableManagedConnection interface introduces another state to the Connection object: *inactive*. A Connection can now be active, closed, and inactive. The connection object enters the inactive state when a corresponding ManagedConnection object is cleaned up. The connection stays inactive until an application component attempts to re-use it. Then the resource adapter calls back to the connection manager to re-associate the connection with an active ManagedConnection object.

Transaction type and connection behavior

All connection usage occurs within the scope of either a global transaction or a local transaction containment (LTC) boundary. Each transaction type places different requirements on connections and impacts connection settings differently.

Connection sharing and reuse

You can only share connections within a global transaction scope (assuming other sharing rules are met). However, you can *serially reuse* connections within an LTC scope. A get/use/close connection pattern followed by another instance of get/use/close (to the same data source or connection factory) enables you to reuse the same connection. See the "Unshareable and shareable connections" on page 574 topic for more details.

JDBC AutoCommit behavior

All JDBC connections, when first obtained through a getConnection() call, contain the setting AutoCommit = TRUE by default. However, different transaction scope and settings can result in changing, or simply overriding, the AutoCommit value.

- If you operate within an LTC and have its resolution-control set to *Application*, then AutoCommit remains *TRUE* unless changed by the application.
- If you operate within an LTC and have its resolution-control set to *ContainerAtBoundary*, then the application should **not** touch the AutoCommit setting. The WebSphere Application Server run time sets the AutoCommit value to *FALSE* before work begins, then commits or rolls back the work as appropriate at the end of the LTC scope.
- If you use a connection within a global transaction, the database ignores the AutoCommit setting so that the transaction service that controls the commit and rollback processing can manage the transaction. This action takes place upon first use of the connection to do work, regardless of the user changing the AutoCommit setting. After the transaction completes, the AutoCommit value returns to the value it had before the first use of the connection. So even if the AutoCommit value is set to *TRUE* before the connection is used in a global transaction, you need not set the value to *FALSE* since the value is ignored by the database. In this example, after the transaction completes, the AutoCommit value of the connection returns to *TRUE*.
- If you use multiple distinct connections within a global transaction, all work is guaranteed to commit or roll back together. This is not the case for a local transaction containment (LTC scope). Within an LTC, work done on one connection commits or rolls back independently from work done on any other connection within the LTC.

One-phase commit and two-phase commit connections

The type and number of resource managers, such as a database server, that must be accessed by an application often determines the application transaction requirements. Consequently each type of resource manager places different requirements on connection behavior.

- A two-phase commit resource manager can support two-phase coordination of a transaction. That support is necessary for transactions that involve other resource managers; these transactions are global transactions. See "Transaction support in WebSphere Application Server" on page 1841 for further explanation.
- A one-phase commit resource manager supports only one-phase transactions, or LTC transactions, in which that resource is the sole participating datastore. Again, see "Transaction support in WebSphere Application Server" on page 1841 for further explanation.

One-phase commit resources are such that work being done on a one phase connection cannot mix with other connections and ensure that the work done on all of the connections completes or fails atomically. The product does not allow more than one one-phase commit connection in a global transaction.

Futhermore, it does not allow a one-phase commit connection in a global transaction with one or more two-phase commit connections. You can coordinate only multiple two-phase commit connections within a global transaction.

WebSphere Application Server provides *last participant support* that enables a single one-phase commit resource to participate in a global transaction with one or more two-phase commit resources.

Note that any time you do multiple getConnection() calls using a resource reference that specifies res-sharing-scope=Unshareable, then you get multiple physical connections. This situation also occurs when res-sharing-scope=Shareable, but the sharing rules are broken. In either case, if you run in a global transaction, ensure the resources involved are enabled for two-phase commit (also sometimes referred to as *JTA Enabled*). Failure to do so results in an XA exception that logs the following message:

```
WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources
 has occurred.
```

Application scoped resources

Use this page to view brief descriptions of the resources that are bundled with your application. You can view individual resource settings by clicking on the resource name.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application_name* > **Application scoped resources**.

Each table row corresponds to a resource that is bundled with your application. Click a resource name or the corresponding provider name to view an administrative console page where you can edit the object configuration settings.

Name:

Specifies the administrative name that was assigned to this resource.

Click this name to view a page where you can edit the configuration settings.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name of the resource.

Data type String

Resource type:

Specifies the type of resource, such as a data source or a J2C connection factory.

Provider:

Specifies the resource provider that supplies the class information for this resource object.

Click the provider name to view a page where you can edit the configuration settings.

Description:

Specifies a text description of the resource.

Cache instances

An application uses a cache instance to store, retrieve, and share data objects within the dynamic cache.

Each cache instance can be configured independently for Java Naming and Directory Interface (JNDI) name, cache size, priority, and disk offload. Objects that are stored in a particular cache instance are not affected by other cache instances. This means that if you store an object named **object_1** with a value of `object_data` in *cache_instance_x*, you can also store an object with the same name, but different value in *cache_instance_y*.

Objects that are stored in a particular cache instance are available to applications on other servers by accessing a cache instance of the same name. The two servers must be within the same replication domain to share data.

There are two types of cache instances, object cache instances and servlet cache instances.

An object cache instance is a location in addition to the default shared dynamic cache where Java 2 Platform, Enterprise Edition (J2EE) applications can store, distribute, and share objects. After configuring object cache instances, you can use the DistributedMap or DistributedObjectCache interfaces in the `com.ibm.websphere.cache` package to programmatically access your cache instances.

See the Reference: Generated API documentation for more information about the DistributedMap or DistributedObjectCache interfaces.

Servlet cache instances are locations in addition to the default dynamic cache where dynamic cache can store, distribute, and share the output and the side effects of an invoked servlet. By configuring a servlet cache instance, your applications have greater flexibility and better tuning of cache resources. The Java Naming and Directory Interface (JNDI) name that is specified for the cache instance in the administrative console maps to the <cache-instance> element in the `cachespec.xml` configuration file. Any <cache-entry> elements that are specified within a <cache-instance> element are created in that specific cache instance. Any <cache-entry> elements that are specified outside of a <cache-instance> element are stored in the default dynamic cache instance.

See “Using servlet cache instances” on page 1982 for more information.

Data access: Resources for learning

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

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

View links to additional information about:

- Programming Specifications
- Container-managed relationships
- Resource adapters
- Miscellaneous articles from the Sun Developer Network and IBM developerWorks Web sites
- Rational Application Developer
- WebSphere Version 5.x Information Center
- IBM Cloudscape
- DB2 database software
- “IBM Informix” on page 584
- Supported hardware, software, and APIs

Programming Specifications
- Enterprise JavaBeans Technology (Source for download of the Enterprise Javabeans 2.1 specification)
- Java™ 2 Platform, Enterprise Edition (J2EE™)
- Java™ Management Extensions (JMX)
- JDBC™ 3.0 API Documentation
- J2EE Connector Architecture Version 1.5 specification
- What's New in the J2EE Connector Architecture 1.5
- What's New in the J2EE Connector Architecture 1.5 (Part 2)

Container-managed relationships

Though this article addresses the EJB 2.0 specification, you still might find parts of it pertinent to your environment.
- Enterprise JavaBeans™ 2.0 Container-Managed Persistence Example

Resource adapters
- The J2EE Connector Architecture Resource Adapter

Miscellaneous articles from the Sun Developer Network and IBM developerWorks Web sites
- Developer Technical Articles & Tips -- Articles: Database Access (Sun Developer Network)
- Sharing connections in WebSphere Application Server V5 (This article is still pertinent to WebSphere Application Server Version 6.0. However, be aware that as of version 6.0, the container-managed authentication type is deprecated.)
- Database authentication in WebSphere Application Server V5 (This article is still pertinent to WebSphere Application Server Version 6.0. However, be aware that the container-managed authentication type is now deprecated.)
- Understanding WebSphere Application Server EJB access intents

Rational Application Developer
- Rational Application Developer for WebSphere Software

WebSphere Version 5.x Information Center
- IBM WebSphere™ Version 5.x Information Center

IBM Cloudscape
- IBM Cloudscape (product section in ibm.com)
- The IBM Cloudscape information center: http://publib.boulder.ibm.com/infocenter/cscv/v10r1/index.jsp.

DB2 database software
- DB2

IBM Informix
- http://www-306.ibm.com/software/data/informix/

Supported hardware, software, and APIs
- Supported hardware, software, and APIs

Configuring data access with scripting

Use these topics to learn about using scripting to configure data access.

This topic contains the following tasks:

Configuring a JDBC provider using scripting

You can configure a JDBC provider using the wsadmin tool and scripting.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

1. There are two ways to perform this task. Perform one of the following:
   - Using the AdminTask object:
     - Using Jacl:

       ```
       $AdminTask createJDBCProvider {-interactive}
       ```

     - Using Jython:

       ```
       AdminTask.createJDBCProvider (['-interactive'])
       ```

   - Using the AdminConfig object:
     a. Identify the parent ID and assign it to the node variable. The following example uses the node configuration object as the parent. You can modify this example to use the cell, cluster, server, or application configuration object as the parent.
        - Using Jacl:

          ```
          set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
          ```

        - Using Jython:

          ```
          node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
          print node
          ```

          Example output:

          ```
          mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
          ```

     b. Identify the required attributes:
        - Using Jacl:

          ```
          $AdminConfig required JDBCProvider
          ```

        - Using Jython:

          ```
          print AdminConfig.required('JDBCProvider')
          ```

          Example output:

          ```
          Attribute     Type
          name     String
          implementationClassName   String
          ```

     c. Set up the required attributes and assign it to the jdbcAttrs variable. You can modify the following example to setup non-required attributes for JDBC provider.

– Using Jacl:

```
set n1 [list name JDBC1]
set implCN [list implementationClassName myclass]
set  jdbcAttrs [list  $n1  $implCN]
```

Example output:

```
{name {JDBC1}} {implementationClassName {myclass}}
```

– Using Jython:

```
n1 = ['name', 'JDBC1']
implCN = ['implementationClassName', 'myclass']
jdbcAttrs = [n1,  implCN]
print jdbcAttrs
```

Example output:

```
[['name', 'JDBC1'], ['implementationClassName', 'myclass']]
```

d. Create a new JDBC provider using node as the parent:

– Using Jacl:

```
$AdminConfig create JDBCProvider $node $jdbcAttrs
```

– Using Jython:

```
AdminConfig.create('JDBCProvider', node, jdbcAttrs)
```

Example output:

```
JDBC1(cells/mycell/nodes/mynode|resources.xml#JDBCProvider_1)
```

2. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
3. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

If you modify the class path or native library path of a JDBC provider: After saving your changes (and synchronizing the node in a network deployment environment), you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

Configuring new data sources using scripting

You can configure new data sources using scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

In WebSphere Application Server, any JDBC driver properties that are required by your database vendor must be set as data source properties. Consult the article Vendor-specific data sources minimum required settings in the *Troubleshooting and support* PDF to see a list of these properties and setting options, ordered by JDBC provider type.

There are two ways to perform this task; use either of the following wsadmin scripting objects:

- AdminTask object
- AdminConfig object

AdminConfig gives you more configuration control than the AdminTask object. When you create a data source using AdminTask, you supply universally required properties only, such as a JNDI name for the data source. (Consult the article “Commands for the JDBCProviderManagement group of the AdminTask object” on page 654 for more information.) Other properties that are required by your JDBC driver are assigned default values by Application Server. You cannot use AdminTask commands to set or edit these properties; you must use AdminConfig commands.

- Using the AdminConfig object to configure a new data source:

1. Identify the parent ID, which is the name and location of the JDBC provider that supports your data source.
   - Using Jacl:
     ```
     set newjdbc [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/]
     ```
   - Using Jython:
     ```
     newjdbc = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/')
     print newjdbc
     ```

   Example output:
   ```
   JDBC1(cells/mycell/nodes/mynode|resources.xml#JDBCProvider_1)
   ```
2. Obtain the required attributes.

   Tip: If the database vendor-required properties (which are referenced in the article "Vendor-specific data sources minimum required settings" on page 687) are not displayed in the resulting list of required attributes, script these properties as data source *custom properties* after you create the data source.
   - Using Jacl:
     ```
     $AdminConfig required DataSource
     ```
   - Using Jython:
     ```
     print AdminConfig.required('DataSource')
     ```

   Example output:
   ```
   Attribute   Type
   name    String
   ```
3. Set up the required attributes.
   - Using Jacl:
     ```
     set name [list name DS1]
     set dsAttrs [list $name]
     ```
   - Using Jython:
     ```
     name = ['name', 'DS1']
     dsAttrs = [name]
     ```
4. Create the data source.
   - Using Jacl:
     ```
     set newds [$AdminConfig create DataSource $newjdbc $dsAttrs]
     ```
   - Using Jython:
     ```
     newds = AdminConfig.create('DataSource', newjdbc, dsAttrs)
     print newds
     ```

   Example output:
   ```
   DS1(cells/mycell/nodes/mynode|resources.xml#DataSource_1)
   ```

- Using the AdminTask object to configure a new data source:
  - Using Jacl:
    ```
    $AdminTask createDatasource {-interactive}
    ```
  - Using Jython:
    ```
    AdminTask.createDatasource (['-interactive'])
    ```
- Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Consult your database documentation to see if the vendor recommends setting additional properties. Script them as data source custom properties. See the article "Configuring new data source custom properties using scripting" on page 595.

Configuring new connection pools using scripting

You can use scripting and the wsadmin tool to configure new connection pools.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newds [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/]
     ```
   - Using Jython:

     ```
     newds = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/')
     ```

   Example output:

   ```
   DS1(cells/mycell/nodes/mynode|resources.xml$DataSource_1)
   ```
2. Creating connection pool:
   - Using Jacl:

     ```
     $AdminConfig create ConnectionPool $newds {}
     ```
   - Using Jython:

     ```
     print AdminConfig.create('ConnectionPool', newds, [])
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#ConnectionPool_1)
   ```
3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Changing connection pool settings with the wsadmin tool

You can use the wsadmin scripting tool to change connection pool settings.

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages. To learn more about the wsadmin tool see Starting the wsadmin scripting client.

To use the wsadmin tool to change connection pool settings:

1. Launch a scripting command. There are several options for you to run scripting commands, ranging from running them interactively to running them in a profile.

   To change connection pool settings, you use the *getAttribute* and *setAttribute* commands, run against the various settings objects.

   For example, to change the connection timeout setting, the commands are:

   ```
   $AdminControl getAttribute $objectname connectionTimeout
   $AdminControl setAttribute $objectname connectionTimeout 200
   ```

   where:
   - $ is a Jacl operator for substituting a variable name with its value
   - getAttribute and setAttribute are the commands
   - connectionTimeout is the object whose value you are resetting
2. For Jacl code examples of each of the connection pool settings, see “Example: Changing connection pool settings with the wsadmin tool” on page 589

Example: Changing connection pool settings with the wsadmin tool

You can use the wsadmin tool to change connection pool settings.

The WebSphere Application Server wsadmin tool provides the ability to run scripts. The WebSphere Application Server only supports the Jacl and Jython scripting languages.

You must start the wsadmin scripting client before you perform any other task using scripting.

Connection Timeout

The Connection timeout can be changed at any time while the pool is active. All connection requests that are waiting for a connection are changed to the new value minus the time already waited, and are returned to the wait state if there are no available connections.

For example, if the connection timeout is changed to 300 seconds and a connection request has waited 100 seconds, the connection request waits for 200 more seconds if no connection becomes available.

```
$AdminControl getAttribute $objectname connectionTimeout
$AdminControl setAttribute $objectname connectionTimeout 200
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Maximum Connections

The maximum connections can be changed at any time except in the case where stuck connection support is active.

If stuck connection support is active, an attempt to change the maximum connections is made. If the attempt fails, an `IllegalState` exception occurs. See the Connection pool advanced settings topic in the *Administering applications and their environment* PDF for more information.

If stuck connection support is not active, the maximum connections are changed to the new value. If the new value is greater than the current value, the number of connections increases to the new value and any requests waiting are notified. If the new value is less than the current value and Aged Timeout or Reap Time is used, the number of connections decreases to the new value depending on pool activity. If agedTimeout or Reap Time are not used, no automatic attempt will be made to reduce the total number of connections. To manually reduce the number of connections to the new maximum connections, use the Mbean function purgePoolContents.

```
$AdminControl getAttribute $objectname maxConnections
$AdminControl setAttribute $objectname maxConnections 200
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Minimum Connections

The minimum connections can be changed at any time

```
$AdminControl getAttribute $objectname minConnections
$AdminControl setAttribute $objectname minConnections 200
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Reap Time

The reap time can be changed at any time. The reap time interval is changed to the new value at the next interval.

```
$AdminControl getAttribute $objectname reapTime
$AdminControl setAttribute $objectname reapTime 30
```

Unused Timeout

The unused timeout can be changed at any time.

```
$AdminControl getAttribute $objectname unusedTimeout
$AdminControl setAttribute $objectname unusedTimeout 900
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Aged Timeout

The Aged Timeout can be changed at any time.

```
$AdminControl getAttribute $objectname agedTimeout
$AdminControl setAttribute $objectname agedTimeout 900
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Purge Policy

The purge policy can be changed at any time.

```
$AdminControl getAttribute $objectname purgePolicy
$AdminControl setAttribute $objectname purgePolicy "Failing Connection Only"
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Surge Protection Support

Surge connection support starts if surgeThreshold is > -1 and surgeCreationInterval is > 0. The surge protection properties can be changed at any time.

```
$AdminControl getAttribute $objectname surgeCreationInterval
$AdminControl setAttribute $objectname surgeCreationInterval 30
$AdminControl getAttribute $objectname surgeThreshold
$AdminControl setAttribute $objectname surgeThreshold 15
```

For more information about this setting, see the Connection pool advanced settings topic in the *Administering applications and their environment* PDF.

Stuck connection Support

An attempt is made to change the stuckTime, stuckTimerTime or stuckThreshold properties. If the attempt fails, an `IllegalState` exception occurs. The pool cannot have any active requests or active connections during this request. For the stuck connection support to start, all three stuck property values must be greater than 0, and the maximum connections value must be > 0.

If the connection pool is stuck, you cannot change the stuck or the maximum connection properties. If you are stuck, there are active connections.

```
$AdminControl getAttribute $objectname stuckTime
$AdminControl setAttribute $objectname stuckTime 30
$AdminControl getAttribute $objectname stuckTimerTime
$AdminControl setAttribute $objectname stuckTimerTime 15
$AdminControl getAttribute $objectname stuckThreshold
$AdminControl setAttribute $objectname stuckThreshold 10
```

For more information about this setting, see the Connection pool advanced settings topic in the *Administering applications and their environment* PDF.

Test Connection Support (This is only supported for a DataSource)

The test connection support starts when the testConnection property is set to true, and the interval is > 0. The test connection properties can be changed at any time.

```
$AdminControl getAttribute $objectname testConnection
$AdminControl setAttribute $objectname testConnection 30
$AdminControl getAttribute $objectname testConnectionInterval
$AdminControl setAttribute $objectname testConnectionInterval 15
```

For more information about this setting, see the Test connection service topic in the *Administering applications and their environment* PDF.

Connection Pool Partition Support

```
$AdminControl invoke $objectname freePoolDistributionTableSize
$AdminControl invoke $objectname numberOfFreePoolPartitions
$AdminControl invoke $objectname numberOfSharedPoolPartitions
$AdminControl invoke $objectname gatherPoolStatisticalData
$AdminControl invoke $objectname enablePoolStatisticalData
```

For more information about this setting, see the Connection pool advanced settings topic in the *Administering applications and their environment* PDF.

Show Pool Information Support

The show pool operations can be changed at any time.

```
$AdminControl invoke $objectname showAllPoolContents
$AdminControl invoke $objectname showPoolContents
$AdminControl invoke $objectname showAllocationHandleList
```

Purge Connection Support

PurgePool can be changed at any time.

```
$AdminControl invoke $objectname purgePoolContents normal
$AdminControl invoke $objectname purgePoolContents immediate
```

Pool Control Support

Pause and resume can be changed at any time.

```
$AdminControl invoke $objectname pause
$AdminControl invoke $objectname resume
```

Example: Connection factory and data source Mbean access using wsadmin

Wsadmin commands are used to access connection factory and data source MBeans. MBeans can retrieve or update properties for a connection factory or data source.

To get a list of J2C connection factory or data source Mbean object names, from the wsadmin command line use one of the following administrative control commands:

```
$AdminControl queryNames *:type=J2CConnectionFactory,*
$AdminControl queryNames *:type=DataSource,*
```

Example output from DataSource query:

```
wsadmin>$AdminControl queryNames *:type=DataSource,*
"WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
system1Node01,JDBCProvider=Cloudscape JDBC Provider,j2eeType=JDBCDataSource,J2EESe
rver=server1,Server=server1,version=6.0.0.0,type=DataSource,mbeanIdentifier=cell
s/system1Node01Cell/nodes/system1Node01/servers/server1/resources.xml#DataSource
_1094760149902,JDBCResource=Cloudscape JDBC Provider,cell=system1Node01Cell"
```

By using the J2C connection factory or data source MBean object name, you can access the MBean. The name follows the `webSphere:name=` expression. In the following example, for the first set, the default data source name is set on variable name. For the second set, the object name is set on variable *objectName*.

Example using the Default Datasource

```
- wsadmin>set name [list Default Datasource]

Default Datasource


- wsadmin>set objectName [$AdminControl queryNames *:name=$name,*]


"WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
system1Node01,JDBCProvider=Cloudscape JDBC Provider,j2eeType=JDBCDataSource,J2EESe
rver=server1,Server=server1,version=6.0.0.0,type=DataSource,mbeanIdentifier=cell
s/system1Node01Cell/nodes/system1Node01/servers/server1/resources.xml#DataSource
_1094760149902,JDBCResource=Cloudscape JDBC Provider,cell=system1Node01Cell"
```

Now you can use the objectName for getting help on attributes and operations available for this mbean.

```
$Help attributes $objectName
$Help operations $objectName
```

To get or set an attribute or to use an operation, use one of the following commands:

```
$AdminControl getAttribute $objectName "attribute name"
$AdminControl setAttribute $objectName "attribute name" value
$AdminControl invoke $objectName "operation"
```

Attribute, operation examples:

```
Get and set a attribute maxConnections (Note, if you get no value, a null value, or
a java.lang.IllegalStateException, the connection factory or data source for this
MBean has not been created. The connection factory or data source is created at first
JNDI lookup. For this example, the Default Datasource needs to be used before get,
set and invoke will work.)

wsadmin>$AdminControl getAttribute $objectName maxConnections
10

wsadmin>$AdminControl setAttribute $objectName maxConnections 20

wsadmin>$AdminControl getAttribute $objectName maxConnections
20

Using invoke

wsadmin>$AdminControl invoke $objectName showPoolContents

PoolManager name:DefaultDatasource
PoolManager object:746354514
Total number of connections: 3 (max/min 20/1, reap/unused/aged 180/1800/0,
connectiontimeout/purge 1800/EntirePool)
```

```
                              (testConnection/inteval false/0, stuck timer/time
/threshold 0/0/0, surge time/connections 0/-1)
Shared Connection information (shared partitions 200)
  No shared connections

Free Connection information (free distribution table/partitions 5/1)
  (2)(0)MCWrapper id 5c6af75b  Managed connection WSRdbManagedConnectionImpl@4b5
a775b  State:STATE_ACTIVE_FREE
  (2)(0)MCWrapper id 3394375a  Managed connection WSRdbManagedConnectionImpl@328
5f75a  State:STATE_ACTIVE_FREE
  (2)(0)MCWrapper id 4795b75a  Managed connection WSRdbManagedConnectionImpl@46a
4b75a  State:STATE_ACTIVE_FREE

  Total number of connection in free pool: 3
UnShared Connection information
  No unshared connections
```

Here is an example Java program using the wsadmin tool to invoke showPoolContents: the Example: using wsadmin to invoke showPoolContents topic in the *Administering applications and their environment* PDF.

Example: Invoking showPoolContents using the wsadmin tool: The following example Java program uses the wsadmin tool to invoke showPoolContents:

```
/**
 * "This sample program is provided AS IS and may be used, executed, copied and modified without royalty payment
 *  by customer (a) for its own instruction and study, (b) in order to develop applications designed to run with
 *  an IBM WebSphere product, either for customer's own internal use or for redistribution by customer, as part
 *  of such an application, in customer's own products. "

 * Product 5724i63, (C) COPYRIGHT International Business Machines Corp., 2004


 *
 * All Rights Reserved * Licensed Materials - Property of IBM
 *
 * This program will display an active mbean object name and the connection pool.
 *
 * To run this program
 *
 *  1. call "%WAS_HOME%/bin/setupCmdLine.bat"
 *
 *  2. "%JAVA_HOME%\bin\java" "%CLIENTSAS%" "-Dwas.install.root=%WAS_HOME%" "-Dwas.repository.root=%CONFIG_ROOT%"
 *     -Dcom.ibm.CORBA.BootstrapHost=%COMPUTERNAME% -classpath "%WAS_CLASSPATH%;%WAS_HOME%\lib\admin.jar;%WAS_HOME%
 *   \lib\wasjmx.jar;." ShowPoolContents DataSource_mbean_name
 */

import java.util.Properties;
import java.util.Set;

import javax.management.*;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;

public class ShowPoolContents {

  private AdminClient adminClient;

  public static void main(String[] args) {
    try {
      String name2 = null;
      String port = null;
      if(args.length <2){
        System.out.println("Enter name for the mbean and port");
        return;
```

```
      } else {
        name2 = args[0];
        port = args[1];
        System.out.println("Searching for name"+name2);
      }
      ShowPoolContents ace = new ShowPoolContents();

      // Create an AdminClient
      ace.createAdminClient(port);

      ObjectName[] cfON = ace.queryMBean("DataSource", null);

      if (cfON.length == 0) {
        System.out.println(" *Error : queryMBean did not find any active mbeans of type DataSource.");
        System.out.println(" At first touch of a DataSource, an mbean will be created.  To touch a
         DataSource, use  " + "getConnection");
        return;
      }
      int selectedObjectName = -1;
      String findName = name2;
      for(int i=0;i " + results);

    } catch (Exception e) {
      System.out.println(" *Exception : " + e.toString());
      e.printStackTrace(System.out);
    }
  }

  private void createAdminClient(String port) {
    // Set up a Properties object for the JMX connector attributes
    Properties connectProps = new Properties();
    connectProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
    connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
    connectProps.setProperty(AdminClient.CONNECTOR_PORT, port);

    // Get an AdminClient based on the connector properties
    try {
      adminClient = AdminClientFactory.createAdminClient(connectProps);
    } catch (ConnectorException e) {
      System.out.println("Exception creating admin client: " + e);
      System.exit(-1);
    }

    System.out.println("Connected to DeploymentManager");
  }

  // helper method to query mbean by type / name
  public ObjectName[] queryMBean(String type, String name) throws Exception {
    String s = "*:";
    if (type != null)
      s += "type=" + type;

    if (name != null)
      s += ",name=" + name;

    s += ",*";

    System.out.println("queryMBean: " + s);
    ObjectName ion = new ObjectName(s);

    Set set = adminClient.queryNames(ion, null);
    Object[] o = set.toArray();
    ObjectName[] on = new ObjectName[o.length];

    for (int i = 0; i <o.length; i++) {
      on[i] = (ObjectName) o[i];
}
```

```
    return on;
  }
}
```

Configuring new data source custom properties using scripting

You can configure new data source custom properties using the wsadmin tool and scripting.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new data source custom property:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set newds [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/]
     ```
   - Using Jython:
     ```
     newds = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/')
     print newds
     ```

   Example output:

   ```
   DS1(cells/mycell/nodes/mynode|resources.xml$DataSource_1)
   ```
2. Get the J2EE resource property set:
   - Using Jacl:
     ```
     set propSet [$AdminConfig showAttribute $newds propertySet]
     ```
   - Using Jython:
     ```
     propSet = AdminConfig.showAttribute(newds, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_8)
   ```
3. Get required attribute:
   - Using Jacl:
     ```
     $AdminConfig required J2EEResourceProperty
     ```
   - Using Jython:
     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```
4. Set up attributes:
   - Using Jacl:
     ```
     set name [list name RP4]
     set rpAttrs [list $name]
     ```
   - Using Jython:
     ```
     name = ['name', 'RP4']
     rpAttrs = [name]
     ```
5. Create a J2EE resource property:
   - Using Jacl:
     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP4(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_8)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new J2CAuthentication data entries using scripting

You can configure new J2CAuthentication data entries with the wsadmin tool and scripting.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new J2CAuthentication data entry:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set security [$AdminConfig getid /Cell:mycell/Security:/]
     ```
   - Using Jython:
     ```
     security = AdminConfig.getid('/Cell:mycell/Security:/')
     print security
     ```
     Example output:
     ```
     (cells/mycell|security.xml#Security_1)
     ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required JAASAuthData
     ```
   - Using Jython:
     ```
     print AdminConfig.required('JAASAuthData')
     ```
     Example output:
     ```
     Attribute       Type
     alias           String
     userId          String
     password        String
     ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set alias [list alias myAlias]
     set userid [list userId myid]
     set password [list password secret]
     set jaasAttrs [list $alias $userid $password]
     ```
     Example output:
     ```
     {alias myAlias} {userId myid} {password secret}
     ```
   - Using Jython:
     ```
     alias = ['alias', 'myAlias']
     userid = ['userId', 'myid']
     password = ['password', 'secret']
     jaasAttrs = [alias, userid, password]
     print jaasAttrs
     ```
     Example output:
     ```
     [['alias', 'myAlias'], ['userId', 'myid'], ['password', 'secret']]
     ```
4. Create JAAS auth data:
   - Using Jacl:
     ```
     $AdminConfig create JAASAuthData $security $jaasAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('JAASAuthData', security, jaasAttrs)
     ```

Example output:

```
(cells/mycell|security.xml#JAASAuthData_2)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new WAS40 data sources using scripting

Use scripting to configure a new WAS40 data source.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjdbc [$AdminConfig getid "/JDBCProvider:Cloudscape JDBC Provider/"]
     ```

   - Using Jython:

     ```
     newjdbc = AdminConfig.getid('/JDBCProvider:Cloudscape JDBC Provider/')
     print newjdbc
     ```

   Example output:

   ```
   JDBC1(cells/mycell/nodes/mynode|resources.xml$JDBCProvider_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WAS40DataSource
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WAS40DataSource')
     ```

   Example output:

   ```
   Attribute    Type
   name    String
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name was4DS1]
     set ds4Attrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'was4DS1']
     ds4Attrs = [name]
     ```

4. Create WAS40DataSource:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig create WAS40DataSource $newjdbc $ds4Attrs]
     ```

   - Using Jython:

     ```
     new40ds = AdminConfig.create('WAS40DataSource', newjdbc, ds4Attrs)
     print new40ds
     ```

   Example output:

   ```
   was4DS1(cells/mycell/nodes/mynode|resources.xml#WAS40DataSource_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new WAS40 connection pools using scripting

You can use scripting to configure a new WAS40 connection pool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new WAS40 connection pool:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig getid /Cell:mycell/Node:mynode/
     Server:server1/JDBCProvider:JDBC1/WAS40DataSource:was4DS1/]
     ```

   - Using Jython:

     ```
     new40ds = AdminConfig.getid('/Cell:mycell/Node:mynode/
     Server:server1/JDBCProvider:JDBC1/WAS40DataSource:was4DS1/')
     print new40ds
     ```

     Example output:

     ```
     was4DS1(cells/mycell/nodes/mynodes:resources.xml$WAS40DataSource_1)
     ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WAS40ConnectionPool
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WAS40ConnectionPool')
     ```

     Example output:

     ```
     Attribute       Type
     minimumPoolSize     Integer
     maximumPoolSize     Integer
     connectionTimeout   Integer
     idleTimeout        Integer
     orphanTimeout       Integer
     statementCacheSize  Integer
     ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set mps [list minimumPoolSize 5]
     set minps [list minimumPoolSize 5]
     set maxps [list maximumPoolSize 30]
     set conn [list connectionTimeout 10]
     set idle [list idleTimeout 5]
     set orphan [list orphanTimeout 5]
     set scs [list statementCacheSize 5]
     set 40cpAttrs [list $minps $maxps $conn $idle $orphan $scs]
     ```

     Example output:

     ```
     {minimumPoolSize 5} {maximumPoolSize 30}
     {connectionTimeout 10} {idleTimeout 5}
     {orphanTimeout 5} {statementCacheSize 5}
     ```

   - Using Jython:

     ```
     minps = ['minimumPoolSize', 5]
     maxps = ['maximumPoolSize', 30]
     conn = ['connectionTimeout', 10]
     idle = ['idleTimeout', 5]
     orphan = ['orphanTimeout', 5]
     scs = ['statementCacheSize', 5]
     cpAttrs = [minps, maxps, conn, idle, orphan, scs]
     print cpAttrs
     ```

     Example output:

```
[[minimumPoolSize, 5], [maximumPoolSize, 30],
[connectionTimeout, 10], [idleTimeout, 5],
[orphanTimeout, 5], [statementCacheSize, 5]]
```

4. Create was40 connection pool:
   - Using Jacl:

     ```
     $AdminConfig create WAS40ConnectionPool $new40ds $40cpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('WAS40ConnectionPool', new40ds, 40cpAttrs)
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode:resources.xml#WAS40ConnectionPool_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new WAS40 custom properties using scripting

You can use scripting and the wsadmin tool to configure a new WAS40 custom property.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new WAS40 custom properties:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig getid /Cell:mycell/Node:mynode/
     JDBCProvider:JDBC1/WAS40DataSource:was4DS1/]
     ```

   - Using Jython:

     ```
     new40ds = AdminConfig.getid('/Cell:mycell/Node:mynode/
     JDBCProvider:JDBC1/WAS40DataSource:was4DS1/')
     print new40ds
     ```

   Example output:

   ```
   was4DS1(cells/mycell/nodes/mynodes|resources.xml$WAS40DataSource_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newds propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newds, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_9)
   ```

3. Get required attribute:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute    Type
   name         String
   ```

4. Set up required attributes:
   - Using Jacl:

```
set name [list name RP5]
set rpAttrs [list $name]
```

- Using Jython:

```
name = ['name', 'RP5']
rpAttrs = [name]
```

5. Create J2EE Resource Property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP5(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_9)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new J2C resource adapters using scripting

Use scripting to configure new J2C resource adapters in WebSphere Application Server.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new J2C resource adapter:

1. Identify the parent ID and assign it to the node variable. The following example uses the node configuration object as the parent. You can modify this example to use the cell, cluster, server, or application configuration object as the parent.
   - Using Jacl:

     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2CResourceAdapter
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2CResourceAdapter')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set rarFile c:/currentScript/cicseci.rar
     set option  [list  -rar.name  RAR1]
     ```

   - Using Jython:

     ```
     rarFile = 'c:/currentScript/cicseci.rar'
     option  = '[-rar.name  RAR1]'
     ```

4. Create a resource adapter:

- Using Jacl:

```
set newra [$AdminConfig installResourceAdapter $rarFile mynode $option]
```

- Using Jython:

```
newra = AdminConfig.installResourceAdapter(rarFile, 'mynode', option)
print newra
```

Example output:

```
RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring custom properties for J2C resource adapters using scripting

You can configure custom properties for J2C resource adapters with scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new custom property for a J2C resource adapters:

1. Identify the parent ID and assign it to the newra variable.
   - Using Jacl:

```
set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
```

   - Using Jython:

```
newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
print newra
```

   Example output:

```
RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
```

2. Get the J2EE resource property set:
   - Using Jacl:

```
set propSet [$AdminConfig showAttribute $newra propertySet]
```

   - Using Jython:

```
propSet = AdminConfig.showAttribute(newra, 'propertySet')
print propSet
```

   Example output:

```
(cells/mycell/nodes/mynode|resources.xml#PropertySet_8)
```

3. Identify the required attributes:
   - Using Jacl:

```
$AdminConfig required J2EEResourceProperty
```

   - Using Jython:

```
print AdminConfig.required('J2EEResourceProperty')
```

   Example output:

```
Attribute     Type
name          String
```

4. Set up the required attributes:
   - Using Jacl:

```
set name [list name RP4]
set rpAttrs [list $name]
```

   - Using Jython:

```
name = ['name', 'RP4']
rpAttrs = [name]
```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP4(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_8)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new J2C connection factories using scripting

Use scripting and the wsadmin tool to configure new J2C connection factories.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new J2C connection factory:

1. Identify the parent ID and assign it to the newra variable.
   - Using Jacl:

     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```

   - Using Jython:

     ```
     newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
     print newra
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

2. There are two ways to configure a new J2C connection factory. Perform one of the following:
   - Using the AdminTask object:
     a. List the connection factory interfaces:
        - Using Jacl:

          ```
          $AdminTask listConnectionFactoryInterfaces $newra
          ```

        - Using Jacl:

          ```
          AdminTask.listConnectionFactoryInterfaces(newra)
          ```

          Example output:

          ```
          javax.sql.DataSource
          ```

     b. Create a J2CConnectionFactory:
        - Using Jacl:

          ```
          $AdminTask createJ2CConnectionFactory $newra { -name cf1
          -jndiName eis/cf1 -connectionFactoryInterface
          avax.sql.DataSource
          ```

        - Using Jacl:

          ```
          AdminTask.createJ2CConnectionFactory(newra, '['-name', 'cf1',
          '-jndiName', 'eis/cf1', '-connectionFactoryInterface',
          'avax.sql.DataSource']')
          ```

   - Using the AdminConfig object:
     a. Identify the required attributes:

– Using Jacl:

```
$AdminConfig required J2CConnectionFactory
```

– Using Jython:

```
print AdminConfig.required('J2CConnectionFactory')
```

Example output:

```
Attribute Type
connectionDefinition ConnectionDefinition@
```

b. If your resource adapter is JCA1.5 and you have multiple connection definitions defined, it is required that you specify the ConnectionDefinition attribute. If your resource adapter is JCA1.5 and you have only one connection definition defined, it will be picked up automatically. If your resource adapter is JCA1.0, you do not need to specify the ConnectionDefinition attribute. Perform the following command to list the connection definitions defined by the resource adapter:

– Using Jacl:

```
$AdminConfig list ConnectionDefinition $newra
```

– Using Jython:

```
print AdminConfig.list('ConnectionDefinition', $newra)
```

c. Set up the required attributes:

– Using Jacl:

```
set name [list name J2CCF1]
set j2ccfAttrs [list $name]
set jname [list jndiName eis/j2ccf1]
```

– Using Jython:

```
name = ['name', 'J2CCF1']
j2ccfAttrs = [name]
jname = ['jndiName', eis/j2ccf1]
```

d. If you are specifying the ConnectionDefinition attribute, also set up the following:

– Using Jacl:

```
set cdattr [list connectionDefinition $cd]
```

– Using Jython:

```
cdattr = ['connectionDefinition', $cd]
```

e. Create a J2C connection factory:

– Using Jacl:

```
$AdminConfig create J2CConnectionFactory $newra $j2ccfAttrs
```

– Using Jython:

```
print AdminConfig.create('J2CConnectionFactory', newra, j2ccfAttrs)
```

Example output:

```
J2CCF1(cells/mycell/nodes/mynode|resources.xml#J2CConnectionFactory_1)
```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new J2C authentication data entries using scripting

You can use scripting to configure a new J2C authentication data entry.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new J2C authentication data entry:

1. Identify the parent ID and assign it to the security variable.
   - Using Jacl:
     ```
     set security [$AdminConfig getid /Security:mysecurity/]
     ```
   - Using Jython:
     ```
     security = AdminConfig.getid('/Security:mysecurity/')
     ```
2. Identify the required attributes:
   - Using Jacl:
     ```
     $AdminConfig required JAASAuthData
     ```
   - Using Jython:
     ```
     print AdminConfig.required('JAASAuthData')
     ```

   Example output:
   ```
   Attribute     Type
   alias         String
   userId          String
   password          String
   ```
3. Set up the required attributes:
   - Using Jacl:
     ```
     set alias [list alias myAlias]
     set userid [list userId myid]
     set password [list password secret]
     set jaasAttrs [list $alias $userid $password]
     ```
     Example output:
     ```
     {alias myAlias} {userId myid} {password secret}
     ```
   - Using Jython:
     ```
     alias = ['alias', 'myAlias']
     userid = ['userId', 'myid']
     password = ['password', 'secret']
     jaasAttrs = [alias, userid, password]
     ```
     Example output:
     ```
     [[alias, myAlias], [userId, myid], [password, secret]]
     ```
4. Create JAAS authentication data:
   - Using Jacl:
     ```
     $AdminConfig create JAASAuthData $security $jaasAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('JAASAuthData', security, jaasAttrs)
     ```

   Example output:
   ```
   (cells/mycell/nodes/mynode|resources.xml#JAASAuthData_2)
   ```
5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new J2C activation specifications using scripting

You can configure new J2C activation specifications using scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a J2C activation specifications:

1. Identify the parent ID and assign it to the `newra` variable.

- Using Jacl:

  ```
  set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
  ```

- Using Jython:

  ```
  newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
  print newra
  ```

  Example output:

  ```
  RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
  ```

2. There are two ways to configure a new J2C administrative object. Perform one of the following:
   - Using the AdminTask object:
     a. List the administrative object interfaces:

        Using Jacl:

        ```
        $AdminTask listMessageListenerTypes $newra
        ```

        Using Jython:

        ```
        AdminTask.listMessageListenerTypes(newra)
        ```

        Example output:

        ```
        javax.jms.MessageListener
        ```

     b. Create a J2C administrative object:

        Using Jacl:

        ```
        $AdminTask createJ2CActivationSpec $newra { -name ac1
        -jndiName eis/ac1 -message ListenerType
        javax.jms.MessageListener}
        ```

        Using Jython:

        ```
        AdminTask.createJ2CActivationSpec(newra, ['-name', 'ao1',
        '-jndiName', 'eis/ao1', '-message', 'ListenerType',
        'javax.jms.MessageListener'])
        ```

   - Using the AdminConfig object:
     a. Using Jacl:

        ```
        $AdminConfig required J2CActivationSpec
        ```

        Using Jython:

        ```
        print AdminConfig.required('J2CActivationSpec')
        ```

        Example output:

        ```
        Attribute Type
        activationSpec ActivationSpec@
        ```

     b. If your resource adapter is JCA V1.5 and you have multiple activation specifications defined, it is required that you specify the activation specification attribute. If your resource adapter is JCA V1.5 and you have only one activation specification defined, it will be picked up automatically. If your resource adapter is JCA V1.0, you do not need to specify the activationSpec attribute. Perform the following command to list the activation specifications defined by the resource adapter:

        Using Jacl:

        ```
        $AdminConfig list ActivationSpec $newra
        ```

        Using Jython:

        ```
        print AdminConfig.list('ActivationSpec', $newra)
        ```

     c. Set the administrative object that you need to a variable:

        Using Jacl:

        ```
        set ac ActivationSpecID
        set name [list name J2CAC1]
        set jname [jndiName eis/j2cac1]
        set j2cacAttrs [list $name $jname]
        ```

        Using Jython:

```
ac = ActivationSpecID
name = ['name', 'J2CAC1']
jname = ['jndiName', 'eis/j2cac1']
j2cacAttrs = [name, jname]
```

d. If you are specifying the ActivationSpec attribute, also set up the following:

Using Jacl:

```
set cdcttr [list activationSpec $ac]
```

Using Jython:

```
cdattr = ['activationSpec', ac]
```

e. Create a J2C activation specification object:

Using Jacl:

```
$AdminConfig create J2CActivationSpec $newra $j2cacAttrs
```

Using Jython:

```
print AdminConfig.create('J2CActivationSpec', newra,j2cacAttrs)
```

Example output:

```
J2CAC1(cells/mycell/nodes/mynode|resources.xml#J2CActivationSpec_1)
```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new J2C administrative objects using scripting

You can use scripting and the wsadmin tool to configure new J2C administrative objects.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a J2C administrative object:

1. Identify the parent ID and assign it to the `newra` variable.
   - Using Jacl:

     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```

   - Using Jython:

     ```
     newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
     print newra
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

2. There are two ways to configure a new J2C administrative object. Perform one of the following:
   - Using the AdminTask object:
     a. List the administrative object interfaces:

        Using Jacl:

        ```
        $AdminTask listAdminObjectInterfaces $newra
        ```

        Using Jython:

        ```
        AdminTask.listAdminObjectInterfaces(newra)
        ```

        Example output:

        ```
        com.ibm.test.message.FVTMessageProvider
        ```

     b. Create a J2C administrative object:

        Using Jacl:

        ```
        $AdminTask createJ2CAdminObject $newra { -name ao1 -jndiName eis/ao1
        -adminObjectInterface com.ibm.test.message.FVTMessageProvider }
        ```

Using Jython:

```
AdminTask.createJ2CAdminObject(newra, ['-name', 'ao1', '-jndiName', 'eis/ao1',
'-adminObjectInterface', 'com.ibm.test.message.FVTMessageProvider'])
```

- Using the AdminConfig object:
  a. Using Jacl:

     ```
     $AdminConfig required J2CAdminObject
     ```

     Using Jython:

     ```
     print AdminConfig.required('J2CAdminObject')
     ```

     Example output:

     ```
     Attribute Type
     adminObject AdminObject@
     ```

  b. If your resource adapter is JCA V1.5 and you have multiple administrative objects defined, it is required that you specify the administrative object attribute. If your resource adapter is JCA V1.5 and you have only one administrative object defined, it will be picked up automatically. If your resource adapter is JCA V1.0, you do not need to specify the administrative object attribute. Perform the following command to list the administrative objects defined by the resource adapter:

     Using Jacl:

     ```
     $AdminConfig list AdminObject $newra
     ```

     Using Jython:

     ```
     print AdminConfig.list('AdminObject', $newra)
     ```

  c. Set the administrative objects that you need to a variable:

     Using Jacl:

     ```
     set ao AdminObjectId
     set name [list name J2CAO1]
     set jname [jndiName eis/j2cao1]
     set j2caoAttrs [list $name $jname]
     ```

     Using Jython:

     ```
     ao = AdminObjectId
     name = ['name', 'J2CAO1']
     set jname = ['jndiName', eis/j2cao1]
     j2caoAttrs = [name, jname]
     ```

  d. If you are specifying the AdminObject attribute, also set up the following:

     Using Jacl:

     ```
     set cdattr [list adminObject $ao]
     ```

     Using Jython:

     ```
     cdattr = ['adminObject', ao]
     ```

  e. Create a J2C administrative object:

     Using Jacl:

     ```
     $AdminConfig create J2CAdminObject $newra $j2caoAttrs
     ```

     Using Jython:

     ```
     print AdminConfig.create('J2CAdminObject', newra, j2caoAttrs)
     ```

     Example output:

     ```
     J2CAO1(cells/mycell/nodes/mynode|resources.xml#J2CAdminObject_1)
     ```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Testing data source connections using scripting

You can test connections for data sources with the wsadmin tool and scripting.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to test a data source to ensure a connection to the database.

1. Identify the DataSourceCfgHelper MBean and assign it to the dshelper variable.
   - Using Jacl:

     ```
     set ds [$AdminConfig getid /DataSource:DS1/]
     $AdminControl testConnection $ds
     ```

   - Using Jython:

     ```
     ds = AdminConfig.getid('/DataSource:DS1/')
     AdminControl.testConnection(ds)
     ```

   Example output:

   ```
   WASX7217I: Connection to provided datasource was successful.
   ```

2. Test the connection. The following example invokes the testConnectionToDataSource operation on the MBean, passing in the classname, userid, password, database name, JDBC driver class path, language, and country.
   - Using Jacl:

     ```
     $AdminControl invoke $dshelper testConnectionToDataSource
     "COM.ibm.db2.jdbc.DB2XADataSource db2admin db2admin
     {{databaseName sample}} c:/sqllib/java/db2java.zip en US"
     ```

   - Using Jython:

     ```
     print AdminControl.invoke(dshelper, 'testConnectionToDataSource',
     'COM.ibm.db2.jdbc.DB2XADataSource dbuser1 dbpwd1
     "{{databaseName jtest1}}" c:/sqllib/java12/db \"\" \"\"')
     ```

   Example output:

   ```
   WASX7217I: Connection to provided data source was successful.
   ```

Commands for the EventServiceDBCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the EventServiceDBCommands group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:

configEvent ServiceDB 2DB	The **configEvent ServiceDB 2DB** command creates the event service database and data sources for DB2 on a server or cluster.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/*dbtype* if this parameter is not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventServiceDB2DB {-createDB true -overrideDataSource true -nodeName nodename -serverName servername -jdbcClassPath c:\sqllib\java -dbUser db2inst1 -dbPassword dbpassword -dbHostName hostname -dbPort 50000 }` • Using Jython string: `AdminTask.configEventServiceDB2DB ('[-createDB true -overrideDataSource true -nodeName nodename -serverName servername -jdbcClassPath c:\sqllib\java -dbUser db2inst1 -dbPassword dbpassword -dbHostName hostname -dbPort 50000 ]')` • Using Jython list: `AdminTask.configEventServiceDB2DB(['-createDB', 'true', '-overrideDataSource', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-jdbcClassPath', 'c:\sqllib\java', '-dbUser', 'db2inst1', '-dbPassword', 'dbpassword ', '-dbHostName', 'hostname', '-dbPort', '50000 '])` **Interactive mode example usage:** • Using Jacl: `$AdminTask configEventServiceDB2DB -interactive` • Using Jython string: `AdminTask.configEventServiceDB2DB('[-interactive]')` • Using Jython list: `AdminTask.configEventServiceDB2DB(['-interactive'])`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- jdbcClassPath** The path to the JDBC driver. Specify only the path to the driver file; do not include the file name in the path. This parameter is required. **- dbNodeName** The DB2 node name (this must be 8 characters or less). This node must be already catalogued and configured to communicate with the DB2 server. This parameter must be set if the current machine is configured as a DB2 client and the parameter **createDB** is set to `true`.	

			- dbHostName The host name of the machine where the database server is installed. This parameter is required. **- dbPort** DB2 instance port. The default value is 50 000 if not specified. **- dbName** The database name to be created. The default value is `event` if not specified. **- dbUser** DB2 user ID that has privileges to create and drop the databases. The default value is `db2inst1` if not specified. **- dbPassword** DB2 password. This parameter is required. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/db2 if this parameter is not specified. • Returns: None	

configEvent ServiceDB2 iSeriesDB	The **configEvent ServiceDB2 iSeriesDB** command generates the DDL database scripts, creates the event service database for DB2 iSeries on the native platform, and creates data sources on a server or cluster.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** When this parameter is set to `true`, the command removes any existing event service data source at the specified scope before creating a new one. When this parameter is set to `false`, the command does not create an event service data source at the specified scope if another event service data source is found at the same scope. The default value is `false` if not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventService DB2iSeriesDB {createDB true -overrideDataSource true -nodeName nodename -serverName servername -dbUser db2user -dbPassword dbpassword -nativeJdbcClassPath /myDB2 ClassPath -collection event}` • Using Jython string: `AdminTask.configEventService DB2iSeriesDB('[-createDB true -overrideDataSource true -nodeName nodename -serverName servername -nativeJdbcClass Path /myDB2ClassPath -colle ction event]')` • Using Jython list: `AdminTask.configEventService DB2iSeriesDB(['-createDB', 'true', '-overrideDataSource', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-nativeJdbcClassPath', '/myDB2ClassPath', '-collec tion', 'event'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask configEventService DB2iSeriesDB -interactive` • Using Jython string: `AdminTask.configEventService DB2iSeriesDB('[-interactive]')` • Using Jython list: `AdminTask.configEventService DB2iSeriesDB(['-interactive'])`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- toolboxJdbcClassPath** The path to the IBM Toolbox for Java DB2 JDBC driver. Specify only the path to the driver file; do not include the file name. You must specify either this parameter or the **jdbcClassPath** parameter. **- nativeJdbcClassPath** The path to the DB2 for iSeries native JDBC driver. Specify only the path to the driver file; do not include the file name in the path. You must specify either this parameter or the **toolboxJdbcClassPath** parameter.	

			- dbHostName The host name of the machine where the DB2 for iSeries database server is installed. This parameter is required if you are using the IBM Toolbox for Java DB2 JDBC driver. **- dbName** The DB2 for iSeries database name. The default value is `*LOCAL` if not specified. **- collection** DB2 for iSeries library SQL collection. The maximum length for the collection name is 10 characters. The default value is an empty string if not specified. **- dbUser** DB2 user ID that has privileges to create and drop the databases. This parameter is required. **- dbPassword** Password of the database user ID. This parameter is required. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/db2iseries if this parameter is not specified. • Returns: None	

configEvent ServiceDB 2ZOSDB	The **configEvent ServiceDB 2ZOSDB** command creates the event service database and data sources for DB2 z/OS on a server or cluster.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** When this parameter is set to `true`, the command removes any existing event service data source at the specified scope before creating a new one. When this parameter is set to `false`, the command does not create an event service data source at the specified scope if another event service data source is found at the same scope. The default value is `false` if not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventService DB2ZOSDB {-createDB true -overrideDataSource true -nodeName nodename -server Name servername -jdbcClassPath c:\sqllib\java -dbUser db2user -dbPassword dbpassword -dbHost Name hostname -dbPort 5027 -dbSubSystemName db2zos81 -dbAliasName event -storageGroup sysdeflt -bufferPool4K BP9 -bufferPool8K BP8K9 -bufferPool 16K BP16K9}` `$AdminTask removeEventService DB2ZOSDB {-removeDB true -node Name nodename -serverName server name -dbAliasName event -dbUser db2user -dbPassword dbpassword }` • Using Jython string: `AdminTask.configEventServiceDB2 ZOSDB(["-createDB", "true", "-overrideDataSource", "true", "-nodeName", "nodename", "-server Name", "servername", "-jdbc ClassPath", "c:\sqllib\java", "-dbUser", "db2user", "-dbPassword", "dbpassword", "-dbHostName", "hostname", "-dbPort", "5027", "-dbSubSystemName", "db2zos81", "-dbAliasName", "event", "-storage Group", "sysdeflt", "-bufferPool 4K" "BP9", "-bufferPool8K", "BP8K9", "-bufferPool16K", "BP16K9"])` `AdminTask.removeEventServiceDB2 ZOSDB(["-removeDB", "true", "-nodeName", "nodename", "-server Name", "servername", "-dbAlias Name", "event", "-dbUser", "db2user", "-dbPassword", "dbpass word"])` • Using Jython list: `AdminTask.configEventServiceDB2 ZOSDB('[-createDB true -overrid eDataSource true -nodeName node name -serverName servername -jdbcClassPath c:\sqllib\java -dbUser db2user -dbPassword dbpassword -dbHostName hostname -dbPort 5027 -dbSubSystemName db2zos81 -dbAliasName event -storageGroup sysdeflt -buffer Pool4K BP9 -bufferPool8K BP8K9 -bufferPool16K BP16K9]')` `AdminTask.removeEventServiceDB2 ZOSDB('[-removeDB true -node Name nodename -serverName servername -dbAliasName event -dbUser db2user -dbPassword dbpassword]')`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- jdbcClassPath** The path to the JDBC driver. Specify only the path to the driver file; do not include the file name in the path. This parameter is required. **- dbHostName** The host name of the machine where the database server is installed. This parameter is required.	**Interactive mode example usage:** • Using Jacl: `$AdminTask configEventServ iceDB2ZOSDB -interactive` • Using Jython string: `AdminTask.configEventServ iceDB2ZOSDB('[-interactive]')` • Using Jython list: `AdminTask.configEventServi ceDB2ZOSDB ['-interactive'])`

			- dbPort DB2 for z/OS instance port. The default value is 5027 if not specified. **- dbAliasName** The name of the cataloged database on the DB2 client machine. This parameter is required when the createDB parameter is set to `true`. **- dbSubSystemName** The DB2 z/OS subsystem that you want to create. This parameter is required. **- eventDBName** The event database name that you want to create. The default value is `event` if you do not specify a value. **- eventCatalogDBName** The event catalog database that you want to create. The default value is `eventcat` if you do not specify a value. **- dbDiskSizeInMB** Specify the disk size in MB for the event service database. This value must be at least 10 MB. The default value is 100 MB if not specified.	

			- dbUser DB2 user ID that has privileges to create and drop the databases. This parameter is required. **- dbPassword** Password of the database user ID. This parameter is required. **- storageGroup** The storage group for the event database and the event catalog database. The storage group must already be created and active. **- bufferPool4K** The name of the 4K buffer pool. This buffer pool must be active before the database DDL scripts can be run. **- bufferPool8K** The name of the 8K buffer pool. This buffer pool must be active before the dabase DDL scripts can be run. **- bufferPool16K** The name of the 16K buffer pool. This buffer pool must be active before the database DDL scripts can be run. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/db2zos if this parameter is not specified. • Returns: None	

configEventServiceDerbyDB	The **configEventServiceDerbyDB** command creates the event service database and data sources for Derby on a server or cluster.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** When this parameter is set to `true`, the command removes any existing event service data source at the specified scope before creating a new one. When this parameter is set to `false`, the command does not create an event service data source at the specified scope if another event service data source is found at the same scope. The default value is `false` if not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventServiceDerbyDB {-createDB true -overrideDataSource true -nodeName nodename -serverName servername}` • Using Jython string: `AdminTask.configEventServiceDerbyDB( '[-createDB true -overrideDataSource true -nodeName nodename -serverName servername]')` • Using Jython list: `AdminTask.configEventServiceDerbyDB(['-createDB', 'true', '-overrideDataSource', 'true', '-nodeName', 'nodename', '-serverName', 'servername'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask configEventServiceDerbyDB -interactive` • Using Jython string: `AdminTask.configEventServiceDerbyDB('[-interactive]')` • Using Jython list: `AdminTask.configEventServiceDerbyDB(['-interactive'])`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- dbHostName** The host name of the network Derby database. To create the Derby network data source, specify this parameter and the **dbPort** parameter. To create the Derby local data source, do not specify this parameter and the **dbPort** parameter. **- dbPort** The port number of the network Derby database. To create the Derby network data source, specify this parameter and the **dbHostName** parameter. To create the Derby local data source, do not specify this parameter and the **dbHostName** parameter.	

			- dbName The database name to be created. The default value is `event` if not specified. **- dbUser** The user ID used by the data source for the Derby database authentication. This parameter is optional when the WebSphere domain security is disabled. If you specify this parameter, you also must specify the **dbPassword** parameter. This parameter is required when the WebSphere domain security is enabled. **- dbPassword** The password used by the data source for the Derby database authentication. This parameter is optional when the WebSphere domain security is disabled. If you specify this parameter, you also must specify the **dbUser** parameter. This parameter is required when the WebSphere domain security is enabled. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/derby if this parameter is not specified. • Returns: None	

configEventServiceInformixDB	The **configEventServiceInformixDB** command creates the event service database and data sources for Informix on a server or cluster.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/*dbtype* if this parameter is not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventService InformixDB {-createDB true -overrideDataSource true -nodeName nodename -serverName servername -jdbcClassPath "c:\program files\ibm\informix\jdbc\lib" -dbInformixDir "c:\program files\ibm\informix" -dbUser informix -dbPassword dbpassword -dbHostName hostname -dbPort 1526 -dbServerName ol_server }` • Using Jython string: `AdminTask.configEventService InformixDB('[-createDB true -overrideDataSource true -nodeName nodename -serverName servername -jdbcClassPath "c:\program files\ibm\informix\jdbc\lib" -dbInformixDir "c:\program files\ibm\informix" -dbUser informix -dbPassword dbpassword -dbHostName hostname -dbPort 1526 -dbServerName ol_server]')` • Using Jython list: `AdminTask.configEventService InformixDB(['-createDB', 'true', '-overrideDataSource', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-jdbcClassPath', 'c:\program files\ibm\informix\jdbc\lib', '-dbInformixDir', 'c:\program files\ibm\informix', '-dbUser', 'informix ', '-dbPassword', 'dbpassword', '-dbHostName', 'hostname', '-dbPort', '1526', '-dbServerName', 'ol_server'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask configEventServiceInformixDB -interactive` • Using Jython string: `AdminTask.configEventServiceInformixDB('[-interactive]')` • Using Jython list: `AdminTask.configEventServiceInformixDB(['-interactive'])`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- jdbcClassPath** The path to the JDBC driver. Specify only the path to the driver file; do not include the file name in the path. This parameter is required. **- dbInformixDir** The directory where the Informix database is installed. This parameter must be specified when the parameter **createDB** is set to `true`. This parameter is required.	

			- dbHostName The host name of the machine where the database server is installed. This parameter is required. **- dbServerName** Informix server instance name (for example, `ol_servername`). This parameter is required. **- dbPort** Informix instance port. The default value is 1526 if not specified. **- dbName** The database name to be created. The default value is `event` if not specified. **- dbUser** The Informix database schema user ID that will own the event service database tables. The WebSphere data source uses this user ID to authenticate the Informix database connection. This parameter is required. **- dbPassword** The password of the schema user ID that owns the event service Informix tables. The WebSphere data source uses this password to authenticate the Informix database connection. This parameter is required.	

			- ceiInstancePrefix The command uses the event service instance name to group the database files in a directory with unique names. The default value is `ceiinst1` if not specified. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/informix if this parameter is not specified. • Returns: None	

configEventServiceOracleDB	The **configEventServiceOracleDB** command creates the event service tables and data sources for Oracle on a server or cluster. The command does not create the database; the Oracle SID must already exist.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/*dbtype* if this parameter is not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventServiceOracleDB {-createDB true -overrideDataSource true -nodeName nodename -serverName servername -jdbcClassPath c:\oracle\ora92\jdbc\lib -oracleHome c:\oracle\ora92 -dbUser ceiuser -dbPassword ceipassword -dbHostName hostname -dbPort 1521 -sysUser sys -sysPassword syspassword}` • Using Jython string: `AdminTask.configEventServiceOracleDB('[-createDB true -overrideDataSource true -nodeName nodename -serverName servername -jdbcClassPath c:\oracle\ora92\jdbc\lib -oracleHome c:\oracle\ora92 -dbUser ceiuser -dbPassword ceipassword -dbHostName hostname -dbPort 1521 -sysUser sys -sysPassword syspassword]')` • Using Jython list: `AdminTask.configEventServiceOracleDB(['-createDB', 'true', '-overrideDataSource', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-jdbcClassPath', 'c:\oracle\ora92\jdbc\lib', '-oracleHome', 'c:\oracle\ora92', '-dbUser', 'ceiuser', '-dbPassword', 'ceipassword', '-dbHostName', 'hostname', '-dbPort', '1521', '-sysUser', 'sys', '-sysPassword', 'syspassword'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask configEventServiceOracleDB -interactive` • Using Jython string: `AdminTask.configEventServiceOracleDB('[-interactive]')` • Using Jython list: `AdminTask.configEventServiceOracleDB(['-interactive'])`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- jdbcClassPath** The path to the JDBC driver. Specify only the path to the driver file; do not include the file name in the path. This parameter is required. **- oracleHome** The ORACLE_HOME directory. This parameter must be set when the parameter **createDB** is set to `true`.	

			- dbHostName The host name of the machine where the Oracle database server is installed. The default value is `localhost` if not specified. **- dbPort** Oracle instance port. The default value is 1521 if not specified. **- dbName** The Oracle system identifier (SID). The SID must already exist and must be available for the event service command to create the tables and populate the tables with data. The default value is `orcl` if not specified. **- dbUser** The Oracle schema user ID that will own the event service Oracle tables. The user ID will be created during the database creation; the WebSphere data source uses this user ID to authenticate the Oracle database connection. The default value is `ceiuser` if not specified. **- dbPassword** The password of the schema user ID. The password will be created during the database creation; the WebSphere data source uses this password to authenticate the Oracle database connection. This parameter is required.	

			- sysUser Oracle sys user ID. This must be a user that has SYSDBA privileges. The default value is `sys` if not specified. **- sysPassword** The password for the user specified by the **sysUser** parameter. The default value is an empty string if not specified. **- ceiInstancePrefix** The command uses the event service instance name to group the database files in a directory with unique names. The default value is `ceiinst1` if not specified. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/oracle if this parameter is not specified. • Returns: None	

configEvent ServiceSQL ServerDB	The **configEvent ServiceSQL ServerDB** command creates the event service database and data sources for SQL Server on a server or cluster.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/*dbtype* if this parameter is not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventService SQLServerDB {-createDB true -overrideDataSource true -nodeName nodename -server Name servername -dbUser cei user -dbPassword ceipassword -dbServerName sqlservername -dbHostName hostname -dbPort 1433 -saUser sa -saPassword sapassword}` • Using Jython string: `AdminTask.configEventService SQLServerDB('[-createDB true -overrideDataSource true -nodeName nodename -server Name servername -dbUser cei user -dbPassword ceipassword -dbServerName sqlservername -dbHostName hostname -dbPort 1433 -saUser sa -saPassword sapassword]')` • Using Jython list: `AdminTask.configEventService SQLServerDB(['-createDB', 'true', '-overrideDataSource', 'true', '-nodeName', 'node name', '-serverName', 'server name', '-dbUser', 'ceiuser', '-dbPassword', 'ceipassword', '-dbServerName', 'sqlserver name', '-dbHostName', 'host name', '-dbPort', '1433', '-saUser', 'sa', '-saPassword', 'sapassword'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask configEventServi ceSQLServerDB -interactive` • Using Jython string: `AdminTask.configEventServi ceSQLServerDB('[-interactive]')` • Using Jython list: `AdminTask.configEventService SQLServerDB(['-interactive'])`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- dbServerName** The server name of the SQL Server database. This parameter must be set when the parameter **createDB** is set to `true`. **- dbHostName** The host name of the machine where the SQL Server database is running.	

			- dbPort SQL Server instance port. The default value is 1433 if not specified. **- dbName** The database name to be created. The default value is `event` if not specified. **- dbUser** The SQL Server user ID that will own the event service tables. The default value is `ceiuser` if not specified. **- dbPassword** The password of the SQL Server user ID specified by the **dbUser** parameter. This parameter is required. **- saUser** User ID that has privileges to create and drop databases and users. This parameter is required when the **createDB** parameter is set to `true`. The default value is `sa` if not specified. **- saPassword** The sa password. You must not specify this parameter if the sa user ID does not have a password.	

			- ceiInstancePrefix The command uses the event service instance name to group the database files in a directory with unique names. The default value is `ceiinst1` if not specified. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/sqlserver if this parameter is not specified. • Returns: None	

configEventServiceSybaseDB	The **configEventServiceSybaseDB** command creates the event service database and data sources for Sybase on a server or cluster.	None	• Parameters: **- createDB** The command generates the DDL database scripts and creates the database when this parameter is set to `true`. The command only generates the DDL database scripts when this parameter is set to `false`. To create the database, the current machine must be already configured to run the database commands. The default value is `false` if not specified. **- overrideDataSource** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/*dbtype* if this parameter is not specified. **- nodeName** The name of the node that contains the server where the event service data source should be created. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask configEventService SybaseDB {-createDB true -overrideDataSource true -nodeName nodename -server Name servername -dbUser cei user -dbPassword ceipassword -createLogin true -dbServer Name sybaseservername -dbHostName hostname -dbPort 5000 -saUser sa -saPassword sapassword -firstDeviceNumber 10 -dbCacheSizeInMB 10 -dbDishSizeInMB 100 -dbInstall Dir c:\sybase}` • Using Jython string: `AdminTask.configEventService SybaseDB('[-createDB true -overrideDataSource true -node Name nodename -serverName servername -dbUser ceiuser -dbPassword ceipassword -createLogin true -dbServer Name sybaseservername -dbHost Name hostname -dbPort 5000 -saUser sa -saPassword sapass word -firstDeviceNumber 10 -dbCacheSizeInMB 10 -dbDish SizeInMB 100 -dbInstallDir c:\sybase]')` • Using Jython list: `AdminTask.configEventService SybaseDB(['-createDB', 'true', '-overrideDataSource', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-dbUser', 'ceiuser', '-dbPass word', 'ceipassword', '-create Login', 'true', '-dbServerName', 'sybaseservername', '-dbHost Name', 'hostname', '-dbPort', '5000', '-saUser', 'sa', '-saPassword', 'sapassword', '-firstDeviceNumber', '10', '-dbCacheSizeInMB', '10', '-dbDishSizeInMB', '100', '-dbInstallDir', 'c:\sybase'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask configEventSer viceSybaseDB -interactive` • Using Jython string: `AdminTask.configEventServi ceSybaseDB('[-interactive]')` • Using Jython list: `AdminTask.configEventServi ceSybaseDB(['-interactive'])`

			- serverName The name of the server where the event service data source should be created. If this parameters is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be created. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- jdbcClassPath** The path to the JDBC driver. Specify only the path to the driver file; do not include the file name in the path. This parameter is required. **- dbInstallDir** The directory where the Sybase database is installed. This parameter must be set when the parameter **createDB** is set to `true`.	

			- dbServerName The name of the Sybase server instance. The server is defined in the Sybase configuration. This parameter is required. **- dbHostName** The host name of the machine where Sybase is running. The default value is `localhost` if not specified. **- dbPort** Sybase instance port. The default value is 5000 if not specified. **- dbName** The database name to be created. The default value is `event` if not specified. **- firstDeviceNumber** The event database creates six devices. This parameter identifies the value of the first device number that should be assigned to the new devices. This parameter must be set when the parameter **createDB** is set to `true`. The default value is 10 if not specified. **- dbCacheSizeInMB** The memory cache size will be used for transaction logs. This parameter must be set when the parameter **createDB** is set to `true`. The lowest valid value is 10. The default value is 10 MB if not specified.	

			- dbDiskSizeInMB The database size in MB to be created for the event service. This parameter must be set when the parameter **createDB** is set to `true`. The lowest valid value is 100. The default value is 100 MB if not specified. **- dbUser** The user ID that will own the event service Sybase tables. The WebSphere data source uses this user ID to authenticate the Sybase database connection. The default value is `ceiuser` if not specified. **- dbPassword** The password of the user ID that owns the event service Sybase tables. The WebSphere data source uses this password to authenticate the Sybase database connection. This parameter is required. **- createLogin** The **configEventServiceSybaseDB** command creates the login user ID that will own the event service Sybase tables when this parameter is set to `true`. The command does not create the user ID when the parameter is set to `false`. The default value is `true` if not specified.	

			- saUser The Sybase sa userid that has privileges to create tables and users. The default value is `sa` if not specified. **- saPassword** The password of the Sybase sa user ID. You must not specify this parameter if the sa user ID does not have a password. **- ceiInstancePrefix** The command uses the event service instance name to group the database files in a directory with unique names. The default value is `ceiinst1` if not specified. **- outputScriptDir** Optional database script output directory. When this parameter is specified, the command generates the event service database scripts in the specified directory. If the specified directory does not contain a full path, the command creates the specified directory in *app_server_root*/profiles/*profile_name*/bin. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/sybase if this parameter is not specified. • Returns: None	

removeEventServiceDB2DB	The **removeEventServiceDB2DB** command removes the event service database and data sources for DB2 from a server or cluster.	None	• Parameters: **- removeDB** The command removes the database when this parameter is set to `true` and does not remove the database when set to `false`. To remove the database, the current machine must already configured to run the database commands. **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventServiceDB2DB {-removeDB true -nodeName nodename -serverName servername -dbUser db2inst1 -dbPassword dbpassword }` • Using Jython string: `AdminTask.removeEventServiceDB2DB('[-removeDB true -nodeName nodename -serverName servername -dbUser db2inst1 -dbPassword dbpassword]')` • Using Jython list: `AdminTask.removeEventServiceDB2DB(['-removeDB', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-dbUser', 'db2inst1', '-dbPassword', 'dbpassword'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServiceDB2DB -interactive` • Using Jython string: `AdminTask.removeEventServiceDB2DB('[-interactive]')` • Using Jython list: `AdminTask.removeEventServiceDB2DB(['-interactive'])`

			- dbUser DB2 user ID that has privileges to create and drop the databases. This parameter must be set when the parameter **removeDB** is set to `true`. The default value is `db2inst1` if not specified. **- dbPassword** DB2 password. This parameter must be set when the parameter **removeDB** is set to true. **- dbScriptDir** The directory containing the database scripts generated by the event service database configuration command. If specified, the command will run the scripts in this directory to remove the event service database. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/db2. • Returns: None	

removeEventServiceDB2iSeriesDB	The **removeEventServiceDB2iSeriesDB** command removes the DB2 for iSeries data sources from a server or cluster. The database must be removed manually.	None	• Parameters: **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventServiceDB2iSeriesDB {-nodeName nodename -serverName servername }` • Using Jython string: `AdminTask.removeEventServiceDB2iSeriesDB('[-nodeName nodename -serverName servername]')` • Using Jython list: `AdminTask.removeEventServiceDB2iSeriesDB(['-nodeName', 'nodename', '-serverName', 'servername'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServiceDB2iSeriesDB -interactive` • Using Jython string: `AdminTask.removeEventServiceDB2iSeriesDB('[-interactive]')` • Using Jython list: `AdminTask.removeEventServiceDB2iSeriesDB(['-interactive'])`

removeEventServiceDB2ZOSDB	The **removeEventServiceDB2ZOSDB** command removes the event service database and data sources for DB2 z/OS from a server or cluster.	None	• Parameters: **- removeDB** The command removes the database when this parameter is set to `true` and does not remove the database when set to `false`. To remove the database, the current machine must already configured to run the database commands. **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified. **- clusterName** The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventService DB2ZOSDB {-removeDB true -nodeName nodename -server Name servername -dbUser db2user -dbPassword dbpassword` • Using Jython string: `AdminTask.removeEventService DB2ZOSDB('[-removeDB true -nodeName nodename -server Name servername -dbUser db2user -dbPassword dbpass word]')` • Using Jython list: `AdminTask.removeEventService DB2ZOSDB(['-removeDB', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-dbUser', 'db2user', '-dbPass word', 'dbpassword'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServ iceDB2ZOSDB -interactive` • Using Jython string: `AdminTask.removeEventServi ceDB2ZOSDB('[-interactive]')` • Using Jython list: `AdminTask.removeEventServi ceDB2ZOSDB(['-interactive'])`

			- dbName The DB2 database name. On the DB2 client machine, it is the name of the catalogued database. On the native z/OS machine, it is the name of the database subsystem. This parameter must be set when the parameter **removeDB** is set to `true`. The default value is `event` if not specified. **- dbUser** DB2 user ID that has privileges to create and drop the databases. This parameter must be set when the parameter **removeDB** is set to true. **- dbPassword** DB2 password. This parameter must be set when the parameter **removeDB** is set to true. **- dbScriptDir** The directory containing the database scripts generated by the event service database configuration command. If specified, the command will run the scripts in this directory to remove the event service database. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/db2zos. • Returns: None	

removeEventServiceDerbyDB	The **removeEventServiceDerbyDB** command removes the Event Service database and data source for Derby from a server or cluster.	None	• Parameters: **- removeDB** The command removes the database when this parameter is set to `true` and does not remove the database when set to `false`. To remove the database, the current machine must already configured to run the database commands. **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventServiceDerbyDB {-removeDB true -nodeName nodename -serverName servername}` • Using Jython string: `AdminTask.removeEventServiceDerbyDB('[-removeDB true -nodeName nodename -serverName servername]')` • Using Jython list: `AdminTask.removeEventServiceDerbyDB(['-removeDB', 'true', '-nodeName', 'nodename', '-serverName', 'servername'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServiceDerbyDB -interactive` • Using Jython string: `AdminTask.removeEventServiceDerbyDB( '[-interactive]')` • Using Jython list: `AdminTask.removeEventServiceDerbyDB(['-interactive'])`

			- clusterName The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- dbScriptDir** The directory containing the database scripts generated by the event service database configuration command. If specified, the command will run the scripts in this directory to remove the event service database. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/derby. • Returns: None	

removeEventServiceInforMixDB	The **removeEventServiceInforMixDB** command removes the event service database and data sources for Informix from a server or cluster.	None	• Parameters: **- removeDB** The command removes the database when this parameter is set to `true` and does not remove the database when set to `false`. To remove the database, the current machine must already configured to run the database commands. **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventService InformixDB {-removeDB true -nodeName nodename -server Name servername}` • Using Jython string: `AdminTask.removeEventService InformixDB('[-removeDB true -nodeName nodename -server Name servername]')` • Using Jython list: `AdminTask.removeEventService InformixDB(['-removeDB', 'true', '-nodeName', 'node name', '-serverName', 'servername'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServi ceInformixDB -interactive` • Using Jython string: `AdminTask.removeEventServi ceInformixDB('[-interactive]')` • Using Jython list: `AdminTask.removeEventServi ceInformixDB(['-interactive'])`

			- clusterName The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- dbScriptDir** The directory containing the database scripts generated by the event service database configuration command. If specified, the command will run the scripts in this directory to remove the event service database. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/informix. • Returns: None	

removeEventServiceOracleDB	The **removeEventServiceOracleDB** command removes the event service tables and data sources for Oracle from a server or cluster. The command does not remove the database.	None	• Parameters: **- removeDB** The command removes the event service tables when this parameter is set to `true` and does not remove the tables when set to `false`. **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventServiceOracleDB {-removeDB true -nodeName nodename -serverName servername -sysUser sys -sysPassword syspassword}` • Using Jython string: `AdminTask.removeEventServiceOracleDB('[-removeDB true -nodeName nodename -serverName servername -sysUser sys -sysPassword syspassword]')` • Using Jython list: `AdminTask.removeEventServiceOracleDB(['-removeDB', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-sysUser', 'sys', '-sysPassword', 'syspassword'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServiceOracleDB -interactive` • Using Jython string: `AdminTask.removeEventServiceOracleDB('[-interactive]')` • Using Jython list: `AdminTask.removeEventServiceOracleDB(['-interactive'])`

			- clusterName The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- sysUser** Oracle database sys user ID. The default value is `sys` if not specified. **- sysPassword** The password for the user specified by the sysUser parameter. **- dbScriptDir** The directory containing the database scripts generated by the event service database configuration command. If specified, the command will run the scripts in this directory to remove the event service database. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/oracle. • Returns: None	

removeEventServiceSQLServerDB	The removeEventServiceOra cleDB command removes the event service database and data sources for SQL Server from a server or cluster.	None	• Parameters: **- removeDB** The command removes the database when this parameter is set to `true` and does not remove the database when set to `false`. To remove the database, the current machine must already configured to run the database commands. **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventService SQLServerDB {-removeDB true -nodeName nodename -serverName servername -dbUser ceiuser -saUser sa -saPassword sapassword -dbServerName sqlservername}` • Using Jython string: `AdminTask.removeEventService SQLServerDB('[-removeDB true -nodeName nodename -serverName servername -dbUser ceiuser -saUser sa -saPassword sapassword -dbServerName sqlservername]')` • Using Jython list: `AdminTask.removeEventService SQLServerDB(['-removeDB', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-dbUser', 'ceiuser', '-saUser','sa', '-saPassword', 'sapassword', '-dbServerName', 'sqlservername'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServiceSQLServerDB -interactive` • Using Jython string: `AdminTask.removeEventServiceSQLServerDB('[-interactive]')` • Using Jython list: `AdminTask.removeEventServiceSQLServerDB(['-interactive'])`

			- clusterName The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- dbServerName** The server name of the SQL Server database. This parameter must be set when the parameter **removeDB** is set to `true`. **- dbUser** SQL Server user ID that owns the event service tables. The default value is `ceiuser` if not specified. **- saUser** User ID that has privileges to drop the databases and users. The default value is `sa` if not specified. **- saPassword** The password specified by the **saUser** parameter. This parameter is required when the parameter **removeDB** is set to `true`.	
			- dbScriptDir The directory containing the database scripts generated by the event service database configuration command. If specified, the command will run the scripts in this directory to remove the event service database. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/sqlserver. • Returns: None	

removeEventServiceSybaseDB	The **removeEventServiceSybaseDB** command removes the event service database and data sources for Sybase from a server or cluster.	None	• Parameters: **- removeDB** The command removes the database when this parameter is set to `true` and does not remove the database when set to `false`. To remove the database, the current machine must already configured to run the database commands. **- nodeName** The name of the node that contains the server where the event service data source should be removed. If this parameter is specified, then the **serverName** parameter must be set. You must not specify this parameter if the **clusterName** parameter is specified. **- serverName** The name of the server where the event service data source should be removed. If this parameter is specified without the **nodeName** parameter, the command will use the node name of the current WebSphere profile. You must not specify this parameter if the **clusterName** parameter is specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeEventService SybaseDB {-removeDB true -nodeName nodename -server Name servername -dbUser cei user -removeLogin true -saUser sa -saPassword sapassword -dbServerName sybaseservername}` • Using Jython string: `AdminTask.removeEventService SybaseDB('[-removeDB true -nodeName nodename -server Name servername -dbUser cei user -removeLogin true -sa User sa -saPassword sapass word -dbServerName sybase servername]')` • Using Jython list: `AdminTask.removeEventService SybaseDB(['-removeDB', 'true', '-nodeName', 'nodename', '-serverName', 'servername', '-dbUser', 'ceiuser', '-remove Login', 'true', '-saUser', 'sa', '-saPassword', 'sapass word', '-dbServerName', 'sybaseservername'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeEventServ iceSybaseDB -interactive` • Using Jython string: `AdminTask.removeEventServ iceSybaseDB('[-interactive]')` • Using Jython list: `AdminTask.removeEventServi ceSybaseDB(['-interactive'])`

			- clusterName The name of the cluster where the event service data source should be removed. If this parameter is specified, then the **serverName** and **nodeName** parameters must not be set. You must not specify this parameter if the **serverName** and **nodeName** parameters are specified. **- dropLogin** The **removeEventServiceSybaseDB** command drops the user ID that owns the event service tables when this parameter is set to `true`. The command will not drop the user ID when this parameter is set to `false`. This parameter must be set when the parameter **removeDB** is set to `true`. The default value is `true` if not specified. **- dbServerName** The name of the Sybase server instance. The server is defined in the Sybase configuration. This parameter is required when the **removeDB** parameter is set to `true`.	

			- dbUser The user ID that owns the event service tables. The default value is `ceiuser` if not specified. **- saUser** The Sybase sa userid that has privileges to drop tables and users. The default value is `sa` if not specified. **- saPassword** The password of the Sybase sa user ID. You must not specify this parameter if the sa user ID does not have a password. **- dbScriptDir** The directory containing the database scripts generated by the event service database configuration command. If specified, the command will run the scripts in this directory to remove the event service database. The default database script output directory is *app_server_root*/profiles/*profile*/databases/event/*node*/*server*/dbscripts/sybase. • Returns: None	

Commands for the JDBCProviderManagement group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the JDBCProviderManagement group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:

create Datasource	The **create Datasource** command creates a new data source to access the back end data store. Application components use the data source to access connection instances to your database.	JDBC Provider Object ID - The configuration object of the JDBC provider to which the new data source will belong.	• Parameters: **- name** The name of the data source. (String, required) **- jndiName** The Java Naming and Directory Interface (JNDI) name. (String, required) **- description** The description of the data source. (String, optional) **- category** The category that you can use to classify a group of data sources. (String, optional) **- dataStoreHelperClassName** The name of the DataStoreHelper implementation class that extends the capabilities of the selected JDBC driver implementation class to perform data-specific functions. WebSphere Application Server provides a set of DataStoreHelper implementation classes for each of the JDBC provider drivers it supports. (String, required) **- componentManagedAuthenticationAlias** The alias used for database authentication at run time. This alias is only used when the application resource reference is using res-auth=Application. (String, optional) **- containerManagedPersistence** Specifies if the data source is used for container managed persistence for enterprise beans. The default value is true. (Boolean, optional)	**Interactive mode example usage:** • Using Jacl: `$AdminTask createDatasource {-interactive}` • Using Jython string: `AdminTask.createDatasource ('[-interactive]')` • Using Jython list: `AdminTask.createDatasource (['-interactive'])`

			• **Parameters for step one:** ***configureResourceProperties*** This command step configures the resource properties that are required by the data source. (Required) You can specify the following parameters for this step: **name** The name of the resource property. (String, required) **type** The type of the resource property. (String, required) **value** The value of the resource property. (String, required) • Returns: The object ID of the created data source configuration object.	

createJDBC Provider	The **createJDBC Provider** command creates a new Java database connectivity provider (JDBC) that you can use to connect to a relational database for data access.	None	• Parameters: **- scope** The scope for the new JDBC provider. The default is `none`. (String, required) **- databaseType** The type of database that will be used by the JDBC provider. For example, DB2, Derby, Informix, Oracle, and so on. (String, required) **- providerType** The JDBC provider type that will be used by the JDBC provider. (String, required) **- implementationType** The implementation type for this JDBC provider. Use `Connection pool data source` if your application runs in a single phase or a local transaction. Otherwise, use `XA data source` to run in a global transaction. (String, required) **- name** The name of the JDBC provider. The default is the value from the provider template. (String, optional) **- description** The description for the JDBC provider. (String, optional) **- implementationClassName** Specifies the Java class name for the JDBC driver implementation. (String, optional) **- classpath** Specifies a list of paths or JAR file names that form the location for the resource provider classes. (String, optional)	**Interactive mode example usage:** • Using Jacl: `$AdminTask createJDBCProvider {-interactive}` • Using Jython string: `AdminTask.createJDBCProvider ('[-interactive]')` • Using Jython list: `AdminTask.createJDBCProvider (['-interactive'])`

			- nativePath Specifies a list of paths that form the location for the resource provider native libraries. (String, optional) • Returns: The ID of the JDBC provider object that you created.	
listDatasources	Use the **listData sources** command to list data sources that are contained in the specified scope.	None	• Parameters: **- scope** The scope for the data sources that will be listed. The valid values include cell, node, server, or cluster. (String, optional) • Returns: A list of data sources.	**Interactive mode example usage:** • Using Jacl: `$AdminTask listDatasources {-interactive}` • Using Jython string: `AdminTask.listDatasources ('[-interactive]')` • Using Jython list: `AdminTask.listDatasources (['-interactive'])`
listJDBC Providers	The **listJDBC Providers** command lists JDBC providers that are contained in the specified scope.	None	• Parameters: **- scope** The scope for the JDBC providers that will be listed. The valid values include cell, node, server, or cluster. (String, optional) • Returns: A list of JDBC providers.	**Interactive mode example usage:** • Using Jacl: `$AdminTask listJDBCProviders {-interactive}` • Using Jython string: `AdminTask.listJDBCProviders ('[-interactive]')` • Using Jython list: `AdminTask.listJDBCProviders (['-interactive'])`

Using object cache instances

Perform this task so that your application can access dynamic cache object cache instances with the DistributedMap or DistributedObjectCache interfaces.

Before you begin, enable the dynamic cache service. See “Enabling the dynamic cache service” on page 1929 for more information.

Using object cache instances can improve the performance of your application because you can programmatically store and share frequently used objects. By using object cache instances, you also have the necessary control over the dynamic cache when you are running multiple applications in an application server. See “Cache instances” on page 582 for more information.

1. Configure one or more cache instances.
   a. In the administrative console, click **Resources > Cache instances > Object cache instances**.
   b. Specify the scope for the cache instance.

      Cell scope makes the cache instance available to all servers within the cell. Node scope makes the cache instance available to all servers on the particular node. Cluster scope makes the cache instance available to all members in a specified cluster. Server scope makes the cache instance available to only the selected server. You can mix scopes if necessary.
   c. Click **Apply** after changing the scope.
   d. Click **New**.
   e. Enter the Java Naming and Directory Interface (JNDI) name for this cache instance.

This is name that you pass to the InitialContext lookup() method from within your application. For example, `services/cache/instance_one`.

f. Enter or modify other properties as needed.

2. Update your application. To store and retrieve objects in an object cache instance, you need a DistributedMap or DistributedObjectCache reference for the named object cache instance. See "Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache" on page 1968 for more information.

You configured object cache instances that you can access programmatically with the DistributedMap and DistributedObjectCache interfaces.

Administering data access applications

These administrative tasks consist primarily of configuring the objects, or resources, through which applications connect with a backend, and tuning those resources to handle the volume of connection requests.

1. If your application contains Web modules or EJB modules that require access to a backend, configure resources according to your type of EIS:
   - For a relational database, follow the steps outlined in the "Configuring a JDBC provider and data source" on page 685 topic.
   - For a non-relational database, or another type of EIS such as the Customer Information Control System (CICS), you must configure a resource adapter and connection factories. The topic Accessing data using J2EE Connector Architecture connectors provides information on setting up these objects.
2. Configure an authentication alias for the new Web module or EJB module resource only if the application code, rather than WebSphere Application Server, authenticates connections with the backend. This security configuration is called *component-managed* authorization, and is indicated in the application deployment descriptor as `res-auth = Application`.

   Container-managed authorization, which is designated as `res-auth = Container`, indicates that Application Server performs signon for backend connections. The container-managed authentication alias must be specified on the application resource reference. This task can be done during application assembly or deployment, along with mapping the resource reference to a data source or connection factory resource. After application deployment, however, you can alter the container-managed authentication alias using the administrative console. Click **Applications** > **Enterprise Applications** > *application_name*, and select the link to the appropriate mapping page. For example, if you want to alter the alias of an EJB module resource, you might click **1.x CMP bean data sources** or **2.x CMP bean data sources**. For a Web module resource, click **Resource References**.

   Consult the "J2EE connector security" on page 682 topic for detailed reference on resource authentication.
3. If your application contains a client module that requires data access, see Configuring data access for application clients. In this single configuration process, you can define authentication data for either component-managed or container-managed signon.
4. Specify connection pool settings.
5. Test a connection to the new data source. See the article "Test connection service" on page 748 for information on the available methods for testing connections. This article also addresses important data source settings that can affect the accuracy of your test connection results.
6. Gather connection pool statistics by activating the "JDBC connection pool counters" on page 2118 or the "J2C connection pool counters" on page 2121. Alternatively, you can use Performance Monitoring Infrastructure (PMI) method calls to gather connection statistics .
7. Tune the resources to manage connection volume. Consult the topic "Tuning parameters for data access resources" on page 771.

Installing a Resource Adapter Archive (RAR) file

WebSphere Application Server uses the classes and other code that comprise a Resource Adapter Archive (RAR) file to support the resource adapters that you configure.

A RAR file, which is often called a J2EE Connector Architecture (JCA) connector, must comply with the JCA Specification. Meet these requirements by using a supported assembly tool (as described in the Assembly tools article) to assemble a collection of Java archive (JAR) files, other runnable components, utility classes, and so on, into a deployable RAR file. Then you are ready to install your RAR file in Application Server.

A RAR file provides the classes and other code to support a resource adapter for access to a specific enterprise information system (EIS), such as the Customer Information Control System (CICS). Hence you configure resource adapters for an EIS only after you install the appropriate RAR file.

1. Click **Resources**.
2. Click **Resource Adapters**.
3. Click **Install RAR**. The Install RAR button opens a dialog for both installing a RAR file and configuring an associated resource adapter. Only click **New** if you want to configure a new resource adapter for a previously installed RAR file.

   Limitation: When installing a RAR file using this dialog, the scope you define on the Resource Adapters page has no effect on where the RAR file is installed. You can install RAR files only at the *node* level, which you specify on the Install RAR page.
4. Browse to find the appropriate RAR file.
   - If your RAR file is located on your local workstation, select **Local path** and browse to find the file.
   - If your RAR file is located on your server, select **Server path** and specify the fully qualified path to the file.
5. Click **Next**.
6. Enter the resource adapter name and any other properties needed under *General Properties*. If you install a JCA resource adapter that includes *Native path* elements, consider the following: If you have more than one native path element, and one of the native libraries (native library A) is dependent on another library (native library B), then you must copy native library B to a *system* directory. Because of limitations on Windows NT and most Unix platforms, an attempt to load a native library does not look in the current directory.
7. Click **OK**.

Installing resource adapters within applications

1. Assemble an application with resource adapter archive (RAR) modules in it. See Assembling applications.
2. Install the application following the steps in Installing a new application. In the **Map modules to servers** step, specify target servers or clusters for each RAR file. Be sure to map all other modules that use the resource adapters defined in the RAR modules to the same targets. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (`plugin-cfg.xml`) for each Web server is generated based on the applications that are routed through it.

   Note: When installing a RAR file onto a server, WebSphere Application Server looks for the manifest (MANIFEST.MF) for the connector module. It looks first in the *connectorModule.jar* file for the RAR file and loads the manifest from the *_connectorModule.jar* file. If the class path entry is in the manifest from the *connectorModule.jar* file, then the RAR uses that class path.

To ensure that the installed connector module finds the classes and resources that it needs, check the **Class path** setting for the RAR using the console. For more information, see "Resource Adapter settings" on page 663 and "WebSphere relational resource adapter settings" on page 558

3. Click **Finish** > **Save** to save the changes.
4. Create connection factories for the newly installed application.
   a. Open the administrative console.
   b. Click **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories**.
   c. Click **New** to create a new connection factory, or click on an existing connection factory to update it.

      If you install a J2C Resource Adapter that includes *Native path* elements, consider the following: If you have more than one native path element, and one of the native libraries (native library A) is dependent on another library (native library B), then you must copy native library B to a *system* directory. Because of limitations on Windows NT and most Unix platforms, an attempt to load a native library does not look in the current directory.

      After you create and save the connection factories, you can modify the resource references defined in various modules of the application and specify the Java Naming and Directory Interface (JNDI) names of the connection factories wherever appropriate.

Note: A given native library can only be loaded one time for each instance of the Java virtual machine (JVM). Because each application has its own classloader, separate applications with embedded RAR files cannot both use the same native library. The second application receives an exception when it tries to load the library.

If any application deployed on the application server uses an embedded RAR file that includes native path elements, then you must always ensure that you shut down the application server cleanly, with no outstanding transactions. If the application server does not shut down cleanly it performs *recovery* upon server restart and loads any required RAR files and native libraries. On completion of recovery, do not attempt any application-related work. Shut down the server and restart it. No further recovery is attempted by the application server on this restart, and normal application processing can proceed.

Install RAR

Use this page to install a RAR file in one of two ways. You can either upload a RAR file from the local file system, or specify an existing RAR file on a server. The RAR file must be installed at the node level, and you can select the node below.

For information on installing a resource adapter, see the article ″Installing a Resource Adapter Archive (RAR) file.″

Related tasks

"Installing a Resource Adapter Archive (RAR) file" on page 660
WebSphere Application Server uses the classes and other code that comprise a Resource Adapter Archive (RAR) file to support the resource adapters that you configure.

Related reference

Administrative console buttons
This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.

Administrative console page features
This topic provides information about the basic elements of an administrative console page, such as the various tabs.

Administrative console preference settings
Use the preference settings to specify how you want information to display on an administrative console page.

Local path:

Specifies the local path of the RAR.

Data type String

Server path:

Specifies the server path where the RAR is located.

Data type String

Scope:

Specifies the scope of the resource adapter. Only applications that are installed within this scope can use this adapter.

Resource Adapters collection

This page contains the list of installed and configured resource adapters and is used to install new resource adapters, create additional configurations of installed resource adapters or delete resource adapter configurations.

A resource adapter is an implementation of the J2EE Connector Architecture (JCA) Specification that provides access for applications to resources outside of the server or provides access for an enterprise information system (EIS) to applications on the server. It can provide application access to resources such as DB2, CICS, SAP and PeopleSoft. It can provide an EIS with the ability to communicate with message-driven beans that are configured on the server. Some resource adapters are provided by IBM; however, third party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of *.rar*. A resource adapter can be provided as a standalone adapter or as part of an application, in which case it is referred to as an embedded adapter. Use this panel to install a standalone resource adapter archive file. Embedded adapters are installed as part of the application installation. This panel can be used to work with either kind of adapter.

To view this administrative console page, click **Resources** > **Resource Adapters** > **Resource Adapters**.

Name:

Specifies the name of the resource adapter.

A string with no spaces meant to be a meaningful text identifier for the resource adapter.

Data type String

Scope:

Specifies the level to which this resource adapter is visible. For general information, see *Administrative console scope settings* in the Related Reference section.

Some considerations that you should keep in mind for this particular panel are:

- Changing the scope enables you to see which resource adapter definitions exist at that level.
- Changing the scope **does not** have any effect on installation. Installations are always done under a scope of **node**, no matter what you set the scope to.
- When you create a new resource adapter from this panel, you must change the scope to what you want it to be **before** clicking **New**.

Resource Adapter settings:

Use this page to specify settings for a Resource Adapter.

A resource adapter is an implementation of the J2EE Connector Architecture (JCA) Specification that provides access for applications to resources outside of the server or provides access for an enterprise information system (EIS) to applications on the server. It can provide application access to resources such as DB2, CICS, SAP and PeopleSoft. It can provide an EIS with the ability to communicate with message driven beans that are configured on the server. Some resource adapters are provided by IBM; however, third party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of *.rar*. A resource adapter can be provided as a standalone adapter or as part of an application, in which case it is referred to as an embedded adapter. Use this panel to install a standalone resource adapter archive file. Embedded adapters are installed as part of the application installation.

To view this administrative console page, click **Resources** >**Resource Adapters** > **Resource Adapters** > *resource_adapter*.

Scope:

Specifies the level to which this resource definition is visible. For general information, see *Administrative console scope settings* in the Related Reference section.

The Scope field is a read only string field that shows where the particular definition for a resource adapter is located. This is set either when the resource adapter is installed (which can only be at the node level) or when a new resource adapter definition is added.

Name:

Specifies the name of the resource adapter definition.

This property is required.

A string with no spaces meant to be a meaningful text identifier for the resource adapter.

Data type String

Description:

Specifies a text description of the resource adapter.

A free-form text string to describe the resource adapter and its purpose.

Data type String

Archive path:

Specifies the path to the RAR file containing the module for this resource adapter.

This property is required.

Data type String

Class path:

Specifies a list of paths or JAR file names which together form the location for the resource adapter classes.

This includes any additional libraries needed by the resource adapter. The resource adapter code base itself is automatically added to the class path, but if anything outside the RAR is needed it can be specified here.

Data type String

Native path:

Specifies a list of paths which forms the location for the resource adapter native libraries.

The resource adapter code base itself is automatically added to the class path, but if anything outside the RAR is needed it can be specified here.

Data type String

ThreadPool Alias:

Specifies the name of a thread pool that is configured in the server that is used by the resource adapter's Work Manager.

If there is no thread pool configured in the server with this name, the default configured thread pool instance, named *Default*, is used. This property is only necessary if this resource adapter uses Work Manager.

This field does not apply for the z/OS platform.

Data type String

Configuring J2EE Connector connection factories in the administrative console

1. Click **Resources**.
2. Click **Resource Adapters**.
3. Select a resource adapter under Resource Adapters.
4. Click **J2C Connection Factories** under Additional Properties .
5. Click **New**.
6. Specify *General Properties* .
7. Select the authentication preference.
8. Select aliases for **component-managed authentication**, **container-managed authentication**, or both. If none are available, or you want to define a different one, click **Apply** > **J2C Authentication Data Entries** under Related Items.
   a. Click **J2C Auth Data Entries** under Related Items.
   b. Click **New**.

c. Specify General Properties.
d. Click **OK**.

9. Click **OK**.
10. Click the J2C connection factory you just created.
11. Under *Additional Properties* click **Connection Pool**.
12. Change any values desired by clicking the property name.
13. Click **OK**.
14. Click **Custom Properties** under *Additional Properties*.
15. Click any property name to change its value. Note that **UserName** and **Password** if present, are overridden by the **component-managed authentication** alias you specified in a previous step.
16. Click **Save**.
17. Restart the Deployment Manager, node agent, and application server for the changes to take effect.

Connection pool settings

Use this page to configure connection pool settings.

This administrative console page is common to a range of resource types: for example, JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection Pool**.

For example: click **Resources** > **JDBC** > **JDBC Providers** > *JDBC_provider* > **Data Sources** > *data_source* > **Connection Pool**. The path for JMS queue connection factories is slightly more involved: **Resources** > **JMS** > **JMS Providers** > **Default Messaging** > **JMS Queue Connection Factory** > *JMS_queue_connection_factory* > **Connection Pool Properties**.

Connection Timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

This value indicates the number of seconds a request for a connection waits when there are no connections available in the free pool and no new connections can be created, usually because the maximum value of connections in the particular connection pool has been reached. For example, if Connection Timeout is set to 300, and the maximum number of connections are all in use, the pool manager waits for 300 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a `ConnectionWaitTimeout` exception. It usually does not make sense to retry the getConnection() method; if a longer wait time is required you should increase the Connection Timeout setting value. If a `ConnectionWaitTimeout` exception is caught by the application, the administrator should review the expected connection pool usage of the application and tune the connection pool and database accordingly.

If the Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection becomes available. This happens when the application completes a transaction and returns a connection to the pool, or when the number of connections falls below the value of Maximum Connections, allowing a new physical connection to be created.

If Maximum Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Maximum Connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown. For example: If the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

Knowing the number of connection pools that can potentially request connections from the backend resource (such as a DB2 database or a CICS server) helps you determine a value to specify for the Maximum Connections property.

For multiple standalone application servers that use the same data source configuration, or J2C connection factory configuration, a separate physical connection pool exists for each server. If you clone these same application servers, WebSphere Application Server implements a separate connection pool for each clone.

All of these connection pools correspond to the same data source or connection factory configuration. Hence all of these connection pools can potentially request connections from the same backend resource, at the same time. The single Maximum Connections value that you set on this console panel applies to every one of these connection pools. Consequently, setting a high Maximum Connections value can result in a load of connection requests that overwhelms your backend resource.

Data type	Integer
Default	10
Range	0 to maximum integer If Max Connections is set to 0, the Connection Timeout value is ignored.

Tip: For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Minimum Connections:

Specifies the minimum number of physical connections to maintain.

If the size of the connection pool is at or below the minimum connection pool size, the Unused Timeout thread does not discard physical connections. However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained. Also, if you set a value for Aged Timeout, connections with an expired age are discarded, regardless of the minimum pool size setting.

For example, if the Minimum Connections value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the Minimum Connections setting.

Data type	Integer
Default	1
Range	0 to max int

Reap Time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The *Reap Time* interval affects the accuracy of the *Unused Timeout* and *Aged Timeout* settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in *Minimum Connections*. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused Timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections exceeds the Minimum Connections setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the Reap Time value. See Reap Time for more information.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged Timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting *Aged Timeout* to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the Reap Timeout value for optimal performance. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that

remains in existence for 1200 seconds (20 minutes) is discarded from the pool. The only exception is if the connection is involved in a transaction when the aged timeout is reached. If it is the connection is closed immediately after the transaction completes.

Note that accuracy of this timeout, as well as performance, are affected by the Reap Time value. See Reap Time for more information.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge Policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**.

Data type	String
Defaults	• EntirePool for J2C connection factories and JMS-related connection factories • EntirePool for WebSphere Version 4.0 data sources • EntirePool for current version data sources that you create through the administrative console • EntirePool for current version data sources that you script through wsadmin AdminConfig commands, invoking JDBC templates that are built into WebSphere Application Server (For information on the command **createUsingTemplate**, see the information center article ″Commands for the AdminConfig object.″) • FailingConnectionOnly for data sources that you script in wsadmin without invoking JDBC templates :

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and issues a stale connection Exception during the next operation on that connection. Subsequent getConnection() requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the stale connection exception is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next getConnection() request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

The connection pretest function attempts to insulate an application from pooled connections that are not valid. When a backend resource, such as a database, goes down, pooled connections that are not valid might exist in the free pool. This is especially true when the purge policy is failingConnectionOnly; in this case, the failing connection is removed from the pool. Depending on the failure, the remaining connections in the pool might not be valid.

Connection pool advanced settings

Use this page to specify connection pooling related settings.

This administrative console page is common to a range of resource types: for example, JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection Pool** > **Advanced connection pool properties**.

For example: click **Resources** > **JDBC** > **JDBC Providers** > *JDBC_provider* > **Data Sources** > *data_source* > **Connection Pool** > **Advanced connection pool properties**. The path for JMS queue connection factories is slightly more involved: **Resources** > **JMS** > **JMS Providers** > **Default Messaging** > **JMS Queue Connection Factory** > *JMS_queue_connection_factory* > **Connection Pool Properties** > **Advanced connection pool properties**.

Properties-shared partitions, free pool partitions, and free pool distribution table size are properties related to reducing the time a thread needs to wait for a synchronization lock.

Related concepts

“Resource adapter” on page 557
A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS).

"Connection factory" on page 563
An application component uses a *connection factory* to access a connection instance, which the component then uses to connect to the underlying enterprise information system (EIS).

"JDBC providers" on page 565
Installed applications use JDBC providers to interact with relational databases.

Number of shared partitions:

Specifies the number of partitions that are created in each of the shared pools.

Partition support is always enabled. The default values of 0 should be used to enable the connection pool to pick the best values for performance. In some cases where large multiprocessor systems are used, adjusting the partition support properties might help performance.

Data type	integer
Default value	0
Range	0 to max int

Number of free pool partitions:

Specifies the number of partitions that are created in each of the free pools.

Data type	integer
Default value	0
Range	0 to max int

Free pool distribution table size:

Determines the distribution of Subject and CRI hash values in the table that indexes connection usage data.

These hash values are used to match connection request credentials with the connections. A free pool distribution table size larger than 1 can yield more efficient distribution of hash values, to help minimize search collisions within the table. Fewer collisions can result in faster retrieval of a connection that matches a request. Use a larger value for free pool distribution table size if your resource receives many incoming requests with varying credentials. Smaller values (1) should be used if the same credentials apply to all incoming requests for the resource. The value of 0 means random distribution.

Data type	integer
Default value	0
Range	0 to max int

Surge threshold:

Specifies the number of connections created before surge protection is activated.

Surge protection is designed to prevent overloading of a data source when too many connections are created at the same time. Surge protection is controlled by two properties, *surge threshold* and *surge creation interval*.

The surge threshold property specifies the number of connections created before surge protection is activated. After you reach the specified number of connections, you enter *surge mode*.

The surge creation interval property specifies the amount of time, in seconds, between the creation of connections when in surge mode.

For example, assume the follow settings:
- maxConnections = 50
- surgeThreshold = 10
- surgeCreationInterval = 30 seconds

If the connection pool receives 15 connection requests, 10 connections are created at about the same time. The 11th connection is created 30 seconds after the first 10 connections. The 12th connection is created 30 seconds after the 11th connection. Connections continue to be created every 30 seconds until there are no more new connections needed or you reach the maxConnections value.

Surge connection support starts if the surge threshold is > -1 and the surge creation interval is > 0. The surge threshold property has a default value of -1, which indicates that it is turned off.

wsadmin examples

```
$AdminControl getAttribute $objectname surgeCreationInterval
$AdminControl setAttribute $objectname surgeCreationInterval 30
$AdminControl getAttribute $objectname surgeThreshold
$AdminControl setAttribute $objectname surgeThreshold 15
```

Data type	integer
Default value	-1
Range	-1 to max int

Surge creation interval:

Specifies the amount of time between connection creates when you are in surge protection mode.

If the number of connections specified in the surge threshold property have been made, each request for a new connection must wait to be created on the surge creation interval. This property has a default value of 20, which indicates that at least 20 seconds should pass between connections being created. Valid values for this property are any positive integer.

Data type	integer
Default value	20
Range	0 to max int

Stuck timer time:

A *stuck* connection is an active connection that is not responding or returning to the connection pool. If the pool appears to be stuck (you have reached the stuck threshold), a resource exception is given to all new connection requests until the pool is unstuck. The stuck timer time property is the interval for the timer. This is how often the connection pool checks for stuck connections. The default value is 5 seconds.

If an attempt to change the stuck time, stuck timer time, or stuck threshold properties using the wsadmin scripting tool fails, an `IllegalState` exception occurs. The pool cannot have any active requests or active connections during this request. For the stuck connection support to start, all three stuck property values must be greater than 0 and maximum connections must be greater than 0.

Also, the stuck timer time, if it is set, must be less than the stuck time value. In fact, it is suggested that the stuck timer time should be one-quarter to one-sixth the value of stuck time so that the connection pool checks for stuck connections 4 to 6 times before a connection is declared stuck. This reduces the likelihood of false positives

wsadmin examples

```
$AdminControl getAttribute $objectname stuckTime
$AdminControl setAttribute $objectname stuckTime 30
$AdminControl getAttribute $objectname stuckTimerTime
$AdminControl setAttribute $objectname stuckTimerTime 15
$AdminControl getAttribute $objectname stuckThreshold
$AdminControl setAttribute $objectname stuckThreshold 10
```

Data type	integer
Default value	5
Range	0 to max int

Stuck time:

A *stuck* connection is an active connection that is not responding or returning to the connection pool. If the pool appears to be stuck (you have reached the stuck threshold), a resource exception is given to all new connection requests until the pool is unstuck. The stuck time property is the interval, in seconds, allowed for a single active connection to be in use to the backend resource before it is considered to be stuck.

Data type	integer
Default value	0
Range	0 to max int

Stuck threshold:

A *stuck* connection is an active connection that is not responding or returning to the connection pool. If the pool appears to be stuck (you have reached the stuck threshold), a resource exception is given to all new connection requests until the pool is unstuck. An application can explicitly catch this exception and continue processing. The pool will continue to periodically check for stuck connections when the number of stuck connections is past the threshold. If the number of stuck connections drops below the stuck threshold, the pool will detect this during its periodic checks and enable the pool to begin servicing requests again. The stuck threshold is the number of connections that need to be considered stuck for the pool to be in stuck mode.

Data type	integer
Default value	0
Range	0 to max int

Connection pool (Version 4) settings

Use this page to create a connection pool for a Version 4.0 data source.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC** > **JDBC Providers** > *JDBC_provider* > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Connection pool properties (version 4)**
- **Resources** > **JDBC** > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Connection pool properties (version 4)**

Scope: Resources such as JDBC providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the cell level, all users in

that cell can look up and use that data source, which is unique within that cell. However, resource property settings are local to each server in the cell. For example, if you define *max connections* to 10, then each server in that cell can have 10 connections.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

For general information, see *Administrative console scope settings* in the Related Reference section.

Data type	String

Minimum Pool Size:

Specifies the minimum number of connections to maintain in the pool.

The minimum pool size can affect the performance of an application. Smaller pools require less overhead when the demand is low because fewer connections are held open to the database. When the demand is high, the first applications experience a slow response because new connections are created if all others in the pool are in use.

Data type	Integer
Default	1
Range	Any non-negative integer.

Maximum Pool Size:

Specifies the maximum number of connections to maintain in the pool.

If the maximum number of connections is reached and all connections are in use, additional requests for a connection wait up to the number of seconds specified as the connection timeout. The maximum pool size can affect the performance of an application. Larger pools require more overhead when demand is high because there are more connections open to the database at peak demand. These connections persist until idled out of the pool. If the maximum value is smaller, longer wait times or possible connection timeout errors during peak times can occur. Ensure that the database can support the maximum number of connections in the application server, in addition to any load that it has outside of the application server.

Data type	Integer
Default	10
Range	Any positive integer

Connection Timeout:

Specifies the maximum number of seconds an application waits for a connection from the pool before timing out and triggering a `ConnectionWaitTimeout` exception. WebSphere Application Server acts on this value only if you set the maximum pool size property, in which case the number of maximum connections serves as a trigger for enforcing the wait timeout property.

Data type	Integer
Units	Seconds
Default	180
Range	Any non-negative integer

Setting this value to 0 disables the connection timeout.

If you accept the default value, Application Server issues the ResourceAllocation exception immediately after the pool manager indicates that the maximum number of connections are in use. If you disable connection timeout, Application Server does not issue an exception. Instead, the pool manager queues subsequent connection requests until it can allocate a connection.

Idle Timeout:

Specifies the maximum number of seconds that an idle (unallocated) connection can remain in the pool before being removed to free resources.

Connections need to idle out of the pool because keeping connections open to the database can cause database memory problems. However, not all connections are idled out of the pool, even if they are older than the Idle Timeout setting. A connection is not idled if removing the connection would cause the pool to shrink below its minimum size. Setting this value to 0 disables the idle timeout.

Data type	Integer
Units	Seconds
Default	1800
Range	Any non-negative integer

Orphan Timeout:

Specifies the maximum number of seconds that an application can hold a connection without using it before the connection returns to the pool

If there is no activity on an allocated connection for longer than the Orphan Timeout setting, the connection is marked for orphaning. After another Orphan Timeout number of seconds, if the connection still has no activity, the connection returns to the pool. If the application tries to use the connection again, it is issued a stale connection exception. Connections that are enlisted in a transaction are not orphaned. Setting this value to 0 disables the orphan timeout.

Data type	Integer
Units	Seconds
Default	1800
Range	Any non-negative integer

Statement Cache Size:

Specifies the number of cached prepared statements to keep per connection.

The largest value you would need to set your cache size to if you do not want any cache discards is determined as follows: for each application that uses this data source on a particular server, add up the number of unique prepared statements (as determined by the *sql* string, concurrency, and the scroll type). This is the maximum number of possible prepared statements that can be cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. This provides better performance. However, because of potential resource limitations, this might not always be possible.

Data type	Integer
Default	10
Range	Any non-negative integer

Auto Connection Cleanup:

Specifies whether or not the connection pooling software automatically closes connections from this data source at the end of a transaction.

The default is *false*, which indicates that when a transaction completes, WebSphere Application Server closes the connection and returns it to the pool. Any use of the connection after the transaction has ended results in a stale connection exception because the connection is closed and has returned to the pool. This mechanism ensures that connections are not held indefinitely by the application. If the value is set to *true*, the connection is not returned to the pool at the end of a transaction. In this case, the application must return the connection to the pool by calling *close()*. If the application does not close the connection, the pool can run out of connections for other applications to use.

Data type	Check box
Default	False (clear)

Configuring connection factories for resource adapters within applications

To access an enterprise information system (EIS), applications require connection factories, which are objects that instantiate resource adapter classes for establishing and maintaining resource connections.

1. Click **Applications**.
2. Click **Install New Application**.
3. Browse to find the appropriate EAR file, which contains an RAR file.
4. Click **Next**.
5. Select **resource ref mapping to a J2C Connection Factory**, then click **Next**.
6. After the application installs, click **Applications**.
7. Select the application just installed.
8. Click **Connector Modules** under Related Items.
9. Select an RAR file name on the Connector Modules page.
10. Click **Resource Adapter** under Additional Properties.
11. Click **J2C Connection Factories** under Additional Properties.
12. Click **New**.
13. Specify General Properties.
14. Select a **Component-managed authentication alias** if any application components with *Application* or *Per connection factory* authentication specified in the resource reference are going to be getting connections from this connection factory using the empty-argument getConnection() method. For resources supporting XA, you can optionally specify an Authentication alias for XA recovery. If a desired alias is not available, or you want to define a different one, click **Apply** > **J2C Authentication Data Entries** under Related Items.
    a. Click **J2C Auth Data Entries** under Related Items.
    b. Click **New**.
    c. Specify General Properties.
    d. Click **OK**.
15. Click **OK**.
16. Click the J2C connection factory you just created.
17. Click **Connection Pool** under Additional Properties .
18. Change any values desired by clicking on the property name.
19. Click **OK**.
20. Click **Custom Properties** under Additional Properties.
21. Click any property name to change its value. Note that **UserName** and **Password** if present, are overridden by the **component-managed authentication** alias you specified in a previous step.
22. Click **Save**.

J2C Connection Factories collection

Use this page to view Java 2 Connector (J2C) connection factories, which represent sets of connection configuration values.

Application components such as enterprise beans have resource reference descriptors that refer to the connection factory, not the resource adapter. The connection factory is really a configuration properties list holder. In addition to the arbitrary set of configuration properties defined by the vendor of the resource adapter, there are several standard configuration properties that apply to the connection factory. These standard properties are used by the Java 2 Connectors connection pool manager in the application server run time and are not known by the vendor-supplied resource adapter code.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource Adapters** > **J2C connection factories**
- **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories**

Name:

Specifies a list of the connection factory display names.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name of this connection factory.

Data type String

Scope:

Specifies the scope of the connection factory. Only applications that are installed within this scope can use this connection factory.

Provider:

Specifies the resource adapter that WebSphere Application Server uses for this connection factory.

Description:

Specifies a text description of this connection factory.

Data type String

Connection factory interface:

Specifies the fully qualified name of the interface that provides the implementation class for the connection factory.

Category:

Specifies a string that you can use to classify or group this connection factory.

Data type String

J2C Connection Factories settings:

Use this page to specify settings for a connection factory.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource Adapters** > **J2C connection factories** > *J2C_connection_factory*
- **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories** > *J2C_connection_factory*

Scope:

Specifies the scope of the resource adapter that connects applications to an enterprise information system (EIS) through this connection factory. Only applications that are installed within this scope can use this connection factory.

Provider:

Specifies the resource adapter that WebSphere Application Server uses for this connection factory.

Provider is displayed in this location only when you create a new connection factory. The list shows all of the existing resource adapters that are defined at the relevant scope. Select one from the list if you want to use an existing resource adapter as Provider.

Create New Provider:

Provides the option of configuring a new resource adapter for the new connection factory.

Create New Provider is displayed only when you create, rather than edit, a connection factory.

Clicking **Create New Provider** triggers the console to display the resource adapter configuration page, where you create a new adapter. After you click **OK** to save your settings, you see the connection factory collection page. Click **New** to define a new connection factory for use with the new resource adapter; the console now displays a configuration page that lists the resource adapter as the new connection factory Provider.

Name:

Specifies a list of connection factory display names.

This is a required property.

Data type String

JNDI Name:

Specifies the JNDI name of this connection factory.

For example, the name could be *eis/myECIConnection*.

After you set this value, save it and restart the server. You can see this string when you run the *dumpNameSpace* tool. This is a required property. If you do not specify a JNDI name, it is filled in by default using the Name field.

Data type String
Default eis/*display name*

Description:

Specifies a text description of this connection factory.

Data type String

Connection Factory Interface:

Specifies the fully qualified name of the Connection Factory Interfaces supported by the resource adapter.

This is a required property. For new objects, the list of available classes is provided by the resource adapter in a drop-down list. After you create the connection factory, the field is a read only text field.

Data type Drop-down list or text

Category:

Specifies a string that you can use to classify or group this connection factory.

Data type String

Component-managed Authentication Alias:

Specifies authentication data for component-managed signon to the resource.

Select an alias from the list.

To define a new alias that is not displayed in the list:
- Click **Apply**. Under Related Items, you now see a listing for J2EE Connector Architecture (J2C) authentication data entries.
- Click **J2EE Connector Architecture (J2C) authentication data entries**.
- Click **New**.
- Define an alias.
- Click **OK**. The console now displays an alias collection page. This page contains a table that lists all of your configured aliases. Before the table, this page also displays the name of your connection factory.
- Click the name of your J2C connection factory. You now see the configuration page for the connection factory.
- Select the new alias in the Component-managed authentication alias list.
- Click **Apply**.

Data type List

Authentication Alias for XA Recovery:

This optional field is used to specify the authentication alias that should be used during XA recovery processing.

If the resource adapter does not support XA transactions, then this field will not be displayed. The default value will come from the selected alias for application authentication (if specified).

Use Component-managed Authentication Alias
Selecting this radio button specifies that the alias set for component-managed authentication is used at XA recovery time.

Data type Radio button

Specify:
Selecting this radio button enables you to choose an authentication alias from a drop-down list of configured aliases.

Data type Radio button

Container-managed Authentication Alias (deprecated):

Specifies authentication data (a string path converted to userid and password) for container-managed signon to the resource.

Note: Beginning with WebSphere Application Server Version 6.0, the container-managed authentication alias is superseded by the specification of a login configuration on the resource-reference mapping at deployment time, for components with *res-auth=Container*.

Select an alias from the list.

To define a new alias that is not displayed in the list:
- Click **Apply**. Under Related Items, you now see a listing for J2EE Connector Architecture (J2C) authentication data entries.
- Click **J2EE Connector Architecture (J2C) authentication data entries**.
- Click **New**.
- Define an alias.
- Click **OK**. The console now displays an alias collection page. This page contains a table that lists all of your configured aliases. Before the table, this page also displays the name of your connection factory.
- Click the name of your J2C connection factory. You now see the configuration page for the connection factory.
- Select the new alias in the Container-managed authentication alias list.
- Click **Apply**.

Data type Pick-list

Authentication Preference (deprecated):

Specifies the authentication mechanisms defined for this connection factory.

Note: Beginning with WebSphere Application Server Version 6.0, the authentication preference is superseded by the combination of the <res-auth> application component deployment descriptor setting and the specification of a login configuration on the resource-reference mapping at deployment time.

This setting specifies which of the authentication mechanisms defined for the corresponding resource adapter applies to this connection factory. Common values, depending on the capabilities of the resource adapter, are: *KERBEROS*, *BASIC_PASSWORD*, and *None*.

If None is chosen, the application component is expected to manage authentication (<res-auth>Application</res-auth>). In this case, the user ID and password are taken from one of the following:
- The component-managed authentication alias
- UserName, Password Custom Properties
- Strings passed on the getConnection method

For example, if two authentication mechanism entries are defined for a resource adapter in the *ra.xml* document:
- <authentication-mechanism-type>BasicPassword</authentication-mechanism-type>
- <authentication-mechanism-type>Kerbv5</authentication-mechanism-type>

the authentication preference specifies the mechanism to use for container-managed authentication. An exception is issued during server startup if a mechanism that is not supported by the resource adapter is selected.

Data type	Pick-list
Default	BASIC_PASSWORD

Mapping-Configuration Alias (deprecated):

Specifies the authentication alias for the Java Authentication and Authorization Service (JAAS) mapping configuration that is used by this connection factory.

Note: Beginning with WebSphere Application Server Version 6.0, the Mapping-Configuration Alias is superseded by the specification of a login configuration on the resource-reference mapping at deployment time, for components with *res-auth=Container*.

Click **Security** > **Secure administration, applications, and infrastructure** > **Java Authentication and Authorization Service** > **Application logins** and select an alias from the table.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations.

Data type	Pick-list

Provider:

Specifies the resource adapter that WebSphere Application Server uses for this connection factory.

Provider is displayed in this location only when you edit the settings of an existing connection factory.

J2C Connection Factory advanced settings:

Use this page to specify settings for a connection factory.

You can access this administrative console page in one of two ways:
- **Resources** > **Resource Adapters** > **J2C connection factories** > *J2C_connection_factory* > **Advanced connection factory properties**
- **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories** > *J2C_connection_factory* > **Advanced connection factory properties**

Related concepts

“Resource adapter” on page 557
A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS).

“Connection factory” on page 563
An application component uses a *connection factory* to access a connection instance, which the component then uses to connect to the underlying enterprise information system (EIS).

“JDBC providers” on page 565
Installed applications use JDBC providers to interact with relational databases.

Log missing transaction contexts:

Specifies whether or not the container logs that there is a missing transaction context when a connection is obtained.

Data type Checkbox

Cached handles:

Specifies whether cached handles (handles held in `inst vars` in a bean) should be tracked by the container.

Data type Checkbox

Connection factory JNDI name tips

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services associate names with locations, services, information, and resources.

Naming services provide name-to-object mappings. Directory services provide information on objects and the search tools required to locate those objects. There are many naming and directory service implementations, and the interfaces to them vary.

Java Naming and Directory Interface (JNDI) provides a common interface that is used to access the various naming and directory services. After you have set this value, saved it, and restarted the server, you should be able to see this string when you invoke name space dump tool.

For WebSphere Application Server specifically, when you create a data source the default JNDI name is set to *jdbc/data_source_name*. When you create a connection factory, its default name is *eis/j2c_connection_factory_name*. You can, of course, override these values by specifying your own.

In addition, if you click the checkbox for the *Use this data source for container managed persistence (CMP)* option when you create the data source, another reference is created with the name of *eis/jndi_name_of_datasource_CMP*. For example, if a data source has a JNDI name of *jdbc/myDatasource*, the CMP JNDI name is *eis/jdbc/myDatasource_CMP*. This name is used internally by CMP and is provided simply for informational purposes.

When creating a connection factory or data source, a JNDI name is given by which the connection factory or data source can be looked up by a component. Preferably an ″indirect″ name with the *java:comp/env* prefix should be used and must be used in future releases. An ″indirect″ name makes any resource-reference data associated with the application available to the connection management runtime, to better manage resources based on the res-auth, res-isolation-level, res-sharing-scope, and res-resolution-control settings.

Though you can use a direct JNDI name, this naming method is deprecated in Version 6 of WebSphere Application Server. Application Server assigns default values to the resource-reference data when you use this method. An informational message, resembling the following, is logged to document the defaults:

```
J2CA0294W: Deprecated usage of direct JNDI lookup of resource jdbc/IOPEntity.
The following default values are used: [Resource-ref settings]

 res-auth:                1 (APPLICATION)
 res-isolation-level:     0 (TRANSACTION_NONE)
 res-sharing-scope:       true (SHAREABLE)
 loginConfigurationName:  null
 loginConfigProperties:   null
[Other attributes]
```

```
 res-resolution-control:   999 (undefined)
isCMP1_x:                 false (not CMP1.x)
isJMS:                 false (not JMS)
```

These default values can lead to unexpected behavior in your resources. For example, an application component (such as a JAAS login module) that accesses a resource with container-managed authentication data might fail to authenticate with the backend resource. Because the res-auth setting is assigned the default level of *Application*, rather than *Container*, the application server cannot find it.

This message can occur when you try using the fully qualified names of resources when looking up resources through Java Naming Directory Interface (JNDI). The J2EE programming model recommends the use of resource references and the local JNDI java:comp/env context. To correct this problem, modify the application to use the preferred J2EE programming model with resource references and the local JNDI java:comp/env context.

J2EE connector security

The J2EE connector architecture defines a standard architecture for connecting the Java 2 Platform, Enterprise Edition (J2EE) to heterogeneous enterprise information systems (EIS). Examples of EIS include Enterprise Resource Planning (ERP), mainframe transaction processing (TP) and database systems.

The connector architecture enables an EIS vendor to provide a standard *resource adapter* for its EIS. A resource adapter is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application. Accessing information in EIS typically requires access control to prevent unauthorized accesses. J2EE applications must authenticate to the EIS to open a connection to it.

The J2EE Connector security architecture is designed to extend the end-to-end security model for J2EE-based applications to include integration with EISs. An application server and an EIS collaborate to ensure the proper authentication of a resource principal, which establishes a connection to an underlying EIS. The connector architecture identifies the following mechanisms as the commonly-supported authentication mechanisms, although other mechanisms can be defined:

- BasicPassword: Basic user-password-based authentication mechanism that is specific to an EIS
- Kerbv5: Kerberos Version 5-based authentication mechanism

Applications define whether to use application-managed signon or container-managed signon in the resource-ref elements in the deployment descriptor. Each resource-ref element describes a single connection factory reference binding. The res-auth element in a resource-ref element, whose value is either `Application` or `Container`, indicates whether the enterprise bean code can perform signon or whether WebSphere Application Server can signon to the resource manager using the principal mapping configuration. The resource-ref element is typically defined at application assembly time with an assembly tool such as the Application Server Toolkit (AST) or Rational Web Developer.assembly tool. The `resource-ref` can also be defined, or redefined, at deployment time.

Application managed signon

To access an EIS system, applications locate a connection factory from the Java Naming and Directory Interface (JNDI) namespace and invoke the getConnection method on that connection factory object. The getConnection method might require a user ID and password argument. A J2EE application can pass in a user ID and password to the getConnection method, which subsequently passes the information to the resource adapter. Specifying a user ID and password in the application code has some security implications, however.

The user ID and password, if coded into the Java source code, are available to developers and testers in the organization. Also, the user ID and password are visible to users if they decompile the Java class.

The user ID and password cannot be changed without first requiring a code change. Alternatively, application code might retrieve sets of user IDs and passwords from persistent storage or from an external service. This approach requires that IT administrators configure and manage a user ID and password using the application-specific mechanism.

WebSphere Application Server supports a component-managed authentication alias to be specified on a resource. This authentication data is common to all references to the resource. On the **Resources > Resource Adapters > J2C connection factories** > *configuration_name* panel, select **Use component-managed authentication alias**.

When `res-auth=Application`, the authentication data is taken from the following elements, in order:

1. The user ID and password that are passed to the getConnection method
2. The component-managed authentication alias in the connection factory or the data source
3. The custom properties user name and password in the data source

The user name and password properties can be initially defined in the resource adapter archive (RAR) file and can also be defined in the administrative console or with wsadmin scripting under custom properties.

Container-managed signon

The user ID and password for the target enterprise information systems (EIS) can be supplied by the application server. WebSphere Application Server provides container-managed signon functionality. The Application Server locates the proper authentication data for the target EIS to enable the client to establish a connection. Application code does not have to provide a user ID and password in the getConnection call when it is configured to use container-managed signon, nor does authentication data have to be common to all references to a resource. WebSphere Application Server uses a Java Authentication and Authorization Service (JAAS) pluggable authentication mechanism to use a pre-configured JAAS login configuration, and LoginModule to map a client security identity and credentials on the running thread to a pre-configured user ID and password.

WebSphere Application Server ships a default many-to-one credential mapping LoginModule module that maps any client identity on the running thread to a preconfigured user ID and password for a specified target EIS. The default mapping module is a special purpose JAAS LoginModule module that returns a PasswordCredential credential that is specified by the configured Java 2 Connector (J2C) authentication data entry. The default mapping LoginModule module performs a table lookup, but does not perform actual authentication. The user ID and password are stored together with an alias in the J2C Authentication data list.

The J2C Authentication data list is located on the Secure administration, applications, and infrastructure panel under **Java Authentication and Authorization Service > J2C Authentication data**. The default principal and credential mapping function is defined by the DefaultPrincipalMapping application JAAS login configuration.

J2C Authentication data that is modified using the administrative console takes effect when the modification is saved into the repository and TestConnection is performed. Also, J2C Authentication data that is modified using wsadmin scripting takes effect when any application is started or restarted for a given WebSphere Application Server server process. J2C Authentication Data modification takes effect by invoking the SecurityAdmin MBean method, updateAuthDataCfg. Set the HashMap parameter to null to enable the Securityadmin MBean to refresh the J2C Authentication data using the latest values in the repository.

Do not modify the DefaultPrincipalMapping login configuration because WebSphere Application Server added performance enhancements to this frequently used default mapping configuration. WebSphere Application Server does not support modifying the DefaultPrincipalMapping configuration, changing the default LoginModule module, or stacking a custom LoginModule module in the configuration.

For most systems, the default method with a many-to-one mapping is sufficient. However, WebSphere Application Server does support custom principal and credential mapping configurations. Custom mapping modules can be added to the application logins JAAS configuration by creating a new JAAS login configuration with a unique name. For example, a custom mapping module can provide one-to-one mapping or Kerberos functionality.

You also can use the WebSphere Application Server administrative console to bind the resource manager connection factory references to one of the configured resource factories. If the value of the res-auth element is `Container` within the deployment descriptor for your application, you must specify the mapping configuration. To specify the mapping configuration, use the **Resource references** link under References on the Applications > Enterprise applications > *application_name* panel. See Mapping resource references to references for additional directions.

J2C mapping modules and mapping properties

Mapping modules are special JAAS login modules that provide principal and credential mapping functionality. You can define and configure custom mapping modules using the administrative console.

You also can define and pass context data to mapping modules by using login options in each JAAS login configuration. In WebSphere Application Server, you also can define context data using mapping properties on each connection factory reference binding.

Login options that are defined under each JAAS login configuration are shared among all resources that use the same JAAS login configuration and mapping modules. Mapping properties that are defined for each connection factory reference binding are used exclusively by that resource reference.

Consider a usage scenario where an external mapping service is used.

For example, you might use the Tivoli Access Manager Global Sign-On (GSO) service. Use the Tivoli Access Manager GSO to locate authentication data for both back-end servers.

You have two EIS servers: DB2 and MQ. The authentication data for DB2 is different from that for MQ, however. Use the login option in a mapping JAAS login configuration to specify the parameters that are required to establish a connection to the Tivoli Access Manager GSO service. Use the mapping properties in a connection factory reference binding to specify which EIS server requires the user ID and password.

For more detailed information about developing a mapping module, see the J2C principal mapping modules article.

Note:

- WebSphere Application Server Version configures container-managed signon under each enterprise application. This action is different than WebSphere Application Server Version 5, which configures container-managed signon for each connection factory.
- The deprecated way of configuration at the **Resources > Resource Adapters > J2C connection factories** > *configuration_name* panel still works in WebSphere Application Server Version 6.1. The advantage to configuring at this level is that the configuration has an application scope and is not visible to other applications. However, the mapping configuration that is defined at this level is visible to other applications.
- The mapping configuration at the connection factory has moved to the resource manager connection factory reference. The mapping login modules that are developed using WebSphere Application Server Version 5 JAAS callback types can be used by the resource manager connection factory reference, but the mapping login modules cannot take advantage of the custom mapping properties feature.
- Connection factory reference binding supports mapping properties, and passes those properties to mapping login modules by way of a new WSMappingPropertiesCallback callback. In addition,

the WSMappingPropertiesCallback callback and the new WSManagedConnectionFactoryCallback callback are defined in the com.ibm.wsspi package. Use the new mapping login modules with the new callback types.

Security of lookups with component managed authentication

External Java clients (stand alone clients or servers from other cells) with Java Naming and Directory Interface (JNDI) access can look up a Java 2 Connector (J2C) resource such as a data source or Java Message Service (JMS) queue. However, they are not permitted to take advantage of the component managed *authentication alias* defined on the resource. This alias is a default value used when the *user* and *password* are not supplied on the getConnection() call. Therefore, if an external client needs to get a connection, it must assume responsibility for the authentication by passing it through arguments on the getConnection() call.

Any client running in the WebSphere Application Server process (such as a Servlet or an enterprise bean) within the same cell that can look up a resource in the JNDI namespace can obtain connections without explicitly providing authentication data on the getConnection() call. In this case, if the component's res-auth setting is **Application**, authentication is taken from the component-managed authentication alias defined on the connection factory. With res-auth set to **Container**, authentication is taken from the login configuration defined on the component's resource-reference. It is important to note that J2C authentication alias is per **cell**. An enterprise bean or Servlet in one application server cannot look up a resource in another server process which is in a different cell, because the alias would not be resolved.

Mapping resource references to references

You can use the WebSphere Application Server administrative console to bind the resource manager connection factory references to one of the configured resource factories.

If the value of the res-auth element is `Container` within the deployment descriptor for your application, you must specify the mapping configuration.

1. Click **Applications** > **Enterprise applications >** *application_name*.
2. Under Additional Properties, select **Resource references**.
3. Select the application module and specify an authentication method for the selected connection factory reference binding. Select either **Use default method** or **Use custom login configuration**. If you select the **Use default method** option, the WebSphere Application Server DefaultPrincipalMapping login configuration is selected. You must select an authentication data alias from the list.
4. After you make a selection, click **Apply** for the configuration to take effect.
5. If you select the **Use custom login configuration** option, you must select a mapping JAAS login configuration from the drop-down list.
6. Click **Apply**. The selected login configuration name and an **Mapping properties** button display in the login configuration field of the particular connection factory reference binding.
7. Click **Mapping properties > New** to specify the properties for your configuration. Click **OK** after specifying the properties on the mapping properties panel.
8. Click **OK** and **Save** on the Resource references panel to save your changes to the master configuration.

Configuring a JDBC provider and data source

For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

Each vendor database requires different JDBC driver implementation classes for JDBC connectivity. Configure a JDBC provider to encapsulate those vendor-specific driver files for use by an application server. The application server also requires data sources to obtain and manage the physical connections between applications and databases. Configure at least one data source for association with each of your JDBC providers.

1. Determine the version of data source that you need according to your code specification level.
   - The Enterprise JavaBean (EJB) 1.0 specification and the Java Servlet 2.2 specification require the Version 4.0 Data source.
   - More advanced releases of these specifications require the current version data source (which is designated in WebSphere Application Server simply as ″Data source,″ with no associated version number).
2. Determine the data source requirements for your application and database.

   Those requirements include JDBC provider types, JDBC driver files, and data source connection properties. Consult the article “Vendor-specific data sources minimum required settings” on page 687 for detailed listings of this information per data source implementation.
3. Create a JDBC provider.

   From the administrative console, see Creating a JDBC provider using the administrative console.

   OR

   Using the wsadmin scripting client, see “Configuring a JDBC provider using scripting” on page 585.

   OR

   Using the Java Management Extensions (JMX) API, see Creating a JDBC provider and data source using the Java Management Extensions API.
4. Create a data source.

   From the administrative console, see Creating a data source using the administrative console.

   OR

   Using the wsadmin scripting client, see “Configuring new data sources using scripting” on page 586. (For V4 data sources, see “Configuring new WAS40 data sources using scripting” on page 597.)

   OR

   Using the JMX API, see Creating a JDBC provider and data source using the Java Management Extensions API.

   Required properties: Different database vendors require different properties for implementations of their JDBC drivers. Set these properties on the WebSphere Application Server data source. Because Application Server contains templates for many vendor JDBC implementations, the administrative console surfaces the required properties and prompts you for them in the **Create data source** wizard. However, if you script your data source configurations, you must consult the article “Vendor-specific data sources minimum required settings” on page 687 for the required properties and settings options.
5. Optional: Configure custom properties. Like the required properties, custom properties for specific vendor JDBC drivers must be set on the Application Server data source. Consult your database documentation for information about available custom properties.
6. Bind resource references to the data source. See the article Data source lookups for enterprise beans and Web modules .
7. Test the connection (for non-container-managed persistence usage). See the “Test connection service” on page 748 article.

If your data source fails, you can find miscellaneous troubleshooting tips by consulting the article“Cannot access a data source” on page 775. If a DB2 data source fails in a 64-bit Application Server environment, consult the technote at the following URL: http://www-1.ibm.com/support/docview.wss?rs=180 &context=SSEQTP&dc=DB520&dc=D600&dc=DB530&dc=D700&dc=DB500&dc=DB540&dc=DB510 &dc=DB550&q1=AMD&uid=swg21212809&loc=en_US&cs=utf-8&lang=en for information on fixing incomplete DB2 installation files.

Vendor-specific data sources minimum required settings

Use this table as an at-a-glance reference of JDBC providers that can be defined for use with WebSphere Application Server Version 6.x, to establish data sources for transacting with relational databases. A list that contains detailed requirements for creating data sources with these providers follows the table. (The list also contains information about JDBC providers that are *deprecated* in WebSphere Application Server Version 6.x.)

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 on Windows, UNIX, or workstation-based LINUX	DB2 Universal JDBC Provider	One phase only	
	DB2 Universal JDBC Provider (XA)	One and two phase	The XA implementation is *not* supported in WebSphere Application Server run on workstation-based LINUX
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 UDB for iSeries	DB2 UDB for iSeries (Native)	One phase only	Recommended for use with WebSphere Application Server **run on iSeries**
	DB2 UDB for iSeries (Native XA)	One and two phase	Recommended for use with WebSphere Application Server **run on iSeries**
	DB2 UDB for iSeries (Toolbox)	One phase only	
	DB2 UDB for iSeries (Toolbox XA)	One and two phase	
	DB2 Universal JDBC Provider (XA)	One and two phase	**-***Only* for use with WebSphere Application Server **run on z/OS** **-**Only driver type 4 is supported **-**Does *not* support Version 4 data sources
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect driver (available from DB2)
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect driver (available from DB2)

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 on z/OS	DB2 for z/OS Local JDBC Provider (RRS), using the Legacy DB2 for OS/390 and z/OS JDBC Driver **Removed support:** WebSphere Application Server v6.1 does not support this provider. Use the DB2 Universal JDBC Driver provider instead. See ″Migrating from the JDBC/SQLJ Driver for OS/390 and z/OS to the DB2 Universal JDBC Driver″ in the Information Management Software for z/OS Solutions Information Center.	One and two phase	*Only* for use with WebSphere Application Server **run on z/OS**
	DB2 Universal JDBC Provider	One phase only	
	DB2 Universal JDBC Provider (XA)	One and two phase	**-**Only driver type 4 is supported in WebSphere Application Server **run on z/OS** **-**Does *not* support Version 4 data sources in WebSphere Application Server **run on z/OS**
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect program (available from DB2)
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect program (available from DB2)

Database type	JDBC Provider	Transaction support	Version and other considerations
Cloudscape Version 10.1.x *Cloudscape v10.1.x provides the Apache Derby runtime.* **JDBC naming requirements:** Because the new Cloudscape code base is the product of the Apache Derby Project, Cloudscape v10.1.x uses Derby JDBC classes. Therefore you must specify Derby JDBC provider names, Derby JDBC driver classes, and Derby data source classes to configure JDBC access to Cloudscape v10.1.x.	Derby JDBC Provider	One phase only	- Not for use in clustered environment: accessible from a single JVM only - Does not support Version 4 data sources
	Derby JDBC Provider (XA)	One and two phase	- Not for use in clustered environment: accessible from a single JVM only - Does not support Version 4 data sources
	Derby Network Server Provider using the Universal JDBC driver *Deprecated in WebSphere Application Server v.6.1*	One phase only	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support XA - Does not support Version 4 data sources
	Derby Network Server using Derby Client provider	One phase only	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support Version 4 data sources - Only for use with Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server
	Derby Network Server using Derby Client provider (XA)	One and two phase	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support Version 4 data sources - Only for use with Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server
Informix	Informix JDBC Provider	One phase only	
	Informix JDBC Provider (XA)	One and two phase	
Sybase	Sybase jConnect for JDBC Provider	One phase only	
	Sybase jConnect for JDBC Provider (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
Oracle	Oracle JDBC Provider	One phase only	
	Oracle JDBC Provider(XA)	One and two phase	
Microsoft SQL Server	DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server	One phase only	- Only for use with the corresponding driver from DataDirect Technologies - Version 3.5, Service pack 2 of the DataDirect ConnectJDBC type 4 driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server (XA)	One and two phase	- Only for use with the corresponding driver from DataDirect Technologies - Version 3.5, Service pack 2 of the DataDirect ConnectJDBC type 4 driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server	One phase only	- Cannot be used outside of WebSphere Application Server environment - Version 3.5, Service pack 2 of the driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server (XA)	One and two phase	- Cannot be used outside of WebSphere Application Server environment - Version 3.5, Service pack 2 of the driver can support access to both MS SQL Server 2005 and MS SQL Server 2000

Detailed data source requirements per JDBC provider and platform

The following list contains the requirements for creating data sources with every JDBC provider type that is supported in WebSphere Application Server Version 6.x. Specific fields are designated for the user and password properties. Inclusion of a property in the list does not imply that you should add it to the data source custom properties list. Rather, inclusion in the list means that a value is *typically* required for that field.

Important: After you determine the type of JDBC provider that suits your application and environment, ensure that you acquire the corresponding JDBC driver at a release level supported by this version of WebSphere Application Server. Consult the WebSphere Application Server prerequisite Web site.

Use these links to find your provider and data source information:

- DB2 on Windows, UNIX, or workstation-based LINUX
- DB2 UDB for iSeries
- DB2 on z/OS, connecting to Application Server on Windows, UNIX, or workstation-based LINUX
- "Cloudscape v10.x" on page 701
  - Formerly known as Derby
  - Comprised of the Derby code base
  - Not supported as a production database with this version of WebSphere Application Server
  - **Cloudscape v10.1.x:** The Cloudscape 10.1.x package that is bundled with WebSphere Application Server is backed by full IBM Quality Assurance (QA).
- Informix
- Sybase
- Oracle
- MS SQL Server

DB2 on Windows, UNIX, or workstation-based LINUX

1. **DB2 Universal JDBC Provider**

   The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2.

   This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

   The DB2 Universal JDBC Driver Provider supports one phase data source:

   com.ibm.db2.jcc.DB2ConnectionPoolDataSource

   Requires JDBC driver files:

   - **db2jcc.jar** After you install DB2, you can find this jar file in the DB2 *java* directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable to point to the **db2jcc.jar** file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable.

     Note: To find out the version of the universal driver you are using, issue this DB2 command:

     ```
     java com.ibm.db2.jcc.DB2Jcc -version
     ```

     The output for the above example is:

     ```
     IBM DB2 JDBC Universal Driver Architecture 2.2.xx
     ```

   - **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in a directory defined by **${UNIVERSAL_JDBC_DRIVER_PATH}**environment variable.
   - **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
     - DB2 Universal
     - DB2 for iSeries
     - DB2 for z/OS
     - SQLDS

     The **db2jcc_license_cisuz.jar** does not ship with Websphere Application Server and should be located in the same directory as the db2jcc.jar file, so that the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** points to both.

```
The classpath for this provider is set as follows:

    <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar </classpath>
    <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
    <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar</classpath>
```

Note: The license jar files are independent of each other; therefore, order does not matter.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **useTransactionRedirect** Configure this property as a data source custom property if your backend uses the Database Partitioning Feature (DPF) of DB2 UDB Version 8.2, fix pack 10, and your partitioning key remains constant throughout a transaction. Activating the property affects how the DB2 Universal JDBC Driver directs each connection request that begins a transaction with DB2. The JDBC driver is triggered to send those connection requests to the DPF node that contains the target data of the first directable statement in the transaction, if such a statement exists. DB2 then directs the SQL statement to different partitions as needed; the transaction proceeds normally from the viewpoint of WebSphere Application Server.

  You can use **useTransactionRedirect** for both **driverType 2** and **driverType 4** data sources. To configure the property, use either the wsadmin scripting tool or the administrative console page. Assign the property the value of `true`. For more information on using the scripting tool or the administrative console for this task, see either the Configuring new data source custom properties using scripting or J2EE resource provider or connection factory custom properties collection chapters in the *Administering applications and their environment* PDF book.

2. **DB2 Universal JDBC Provider (XA)**

   The DB2 Universal JDBC Provider (XA) is an architecture-neutral JDBC provider for distributed and local DB2 access. Whether you use this provider for Java connectivity or Java Native Interface (JNI) based connectivity depends on the version of DB2 you are running. Application Server Version 6.0 minimally requires DB2 8.1 Fix Pack 6. This version of DB2 only supports XA connectivity over the Java Native Interface (JNI) based connectivity (Type 2) driver. In order to use XA connectivity with the Type 4 driver, DB2 8.1 Fix Pack 7 or higher is required.

   The DB2 Universal JDBC Driver (XA) supports two phase transactions and the more advanced data source option offered by Application Server (as opposed to the other option, Version 4 data sources). This driver also allows applications to use both JDBC and SQLJ access.

   The DB2 Universal JDBC Driver Provider supports the two phase data source:

   com.ibm.db2.jcc.DB2XADataSource

   Requires JDBC driver files:

   - `db2jcc.jar` This is the DB2 Universal JDBC Driver JAR file. After you install DB2, you can find this JAR file in the DB2 Java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also specify the fully qualified path of `db2jcc.jar` as the value of the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable:

`${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar`

- `db2jcc_License_cu.jar` This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this JAR file or the next one to gain access to the database. This JAR file ships with WebSphere Application Server in the *WAS_HOME/universalDriver/lib* directory. Class path:

  `${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cu.jar`
- `db2jcc_License_cisuz.jar` This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  Class path:

  `${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cisuz.jar`

  You must use the right license JAR file to access a specific database backend.
- The native files required by the DB2 Universal JDBC Driver in WebSphere Application Server. Use the following library path:

  `${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}`

Requires **`DataStoreHelper`** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 XA driver, set this value to 2. If you want to use Universal JDBC Type 4 XA driver (which requires DB2 8.1 Fix Pack 7 or higher), set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **useTransactionRedirect** Configure this property as a data source custom property if your backend uses the Database Partitioning Feature (DPF) of DB2 UDB Version 8.2, fix pack 10, and your partitioning key remains constant throughout a transaction. Activating the property affects how the DB2 Universal JDBC Driver directs each connection request that begins a transaction with DB2. The JDBC driver is triggered to send those connection requests to the DPF node that contains the target data of the first directable statement in the transaction, if such a statement exists. DB2 then directs the SQL statement to different partitions as needed; the transaction proceeds normally from the viewpoint of WebSphere Application Server.

  You can use **useTransactionRedirect** for both **driverType 2** and **driverType 4** data sources. To configure the property, use either the wsadmin scripting tool or the administrative console page “J2EE resource provider or connection factory custom properties collection” on page 733. Assign the property the value of `true`.

Tip: To find the level of universal driver you are using, issue the following DB2 command:

```
java com.ibm.db2.jcc.DB2Jcc -version
```

example output of the above:

```
IBM DB2 JDBC Universal Driver Architecture 2.2.xx
```

3. **DB2 legacy CLI-based Type 2 JDBC Driver**

Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider instead.

The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers.

DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2DataStoreHelper`

Does not require a valid authentication alias if Application Server is running on the same machine as the database. Otherwise, connectivity through this driver does require an alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider (XA) instead.

The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers.

DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

COM.ibm.db2.jdbc.DB2XADataSource

Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2DataStoreHelper`

Does not require a valid authentication alias if Application Server is running on the same machine as the database. Otherwise, connectivity through this driver does require an alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

DB2 UDB for iSeries

1. **DB2 UDB for iSeries (Native)**

The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2, or later releases.

DB2 UDB for iSeries (Native V5R2 and later) supports one phase data source:

com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource

Requires JDBC driver files: **db2_classes.jar**

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

Does not require an authentication alias.

Requires properties:

- **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

2. **DB2 UDB for iSeries (Native XA)**

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2 or later releases.

   DB2 UDB for iSeries (Native XA - V5R2 and later) supports two phase data source:

   com.ibm.db2.jdbc.app.UDBXADataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

3. **DB2 UDB for iSeries (Toolbox)**

   This JDBC driver, also known as iSeries Toolbox driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

   DB2 UDB for iSeries (Toolbox) supports one phase data source:

   com.ibm.as400.access.AS400JDBCConnectionPoolDataSource

   Requires JDBC driver files: **jt400.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

   Requires properties:

   - **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

4. **DB2 UDB for iSeries (Toolbox XA)**

   This XA compliant JDBC driver, also known as iSeries Toolbox XA compliant driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

   DB2 UDB for iSeries (Toolbox XA) supports two phase data source:

   com.ibm.as400.access.AS400JDBCXADataSource

   Requires JDBC driver files: **jt400.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

   Requires properties:

   - **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

5. **DB2 legacy CLI-based Type 2 JDBC Driver**

   Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider instead.

   The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. This provider is intended for *remote* connections to DB2 running on iSeries; for use with Application Server on Windows, UNIX, or workstation-based LINUX, it therefore requires the DB2 Connect Driver (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

   COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

   Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

6. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider (XA) instead.

   The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. This provider is intended for *remote* connections to DB2 running on iSeries; for use with Application Server on Windows, UNIX, or workstation-based LINUX, it therefore requires the DB2 Connect Driver (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

   COM.ibm.db2.jdbc.DB2XADataSource

   Requires JDBC driver files: **db2java.zip**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2DataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

7. **DB2 UDB for iSeries (Native - Version 5 Release 1 and earlier)** -- Deprecated

   This JDBC provider is deprecated because it corresponds to a version of the iSeries operating system that WebSphere Application Server Version 6.x does not support. You must now use iSeries V5R2 or a later release of the iSeries operating system, for which the WebSphere Application Server administrative console lists one native iSeries DB2 non-XA provider: DB2 UDB for iSeries (Native).

   The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native V5R1 and earlier) supports one phase data source:

   com.ibm.db2.jdbc.app.DB2StdConnectionPoolDataSource

   Requires JDBC driver files: **db2_classes.jar**

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

Does not require an authentication alias.

Requires properties:

- **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

8. **DB2 UDB for iSeries (Native XA - Version 5 Release 1 and earlier)** -- Deprecated

   This JDBC provider is deprecated because it corresponds to a version of the iSeries operating system that WebSphere Application Server Version 6.x does not support. You must now use iSeries V5R2 or a later release of the iSeries operating system, for which the administrative console lists one native iSeries DB2 XA provider: DB2 UDB for iSeries (Native XA).

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native XA - V5R1 and earlier) supports two phase data source:

   com.ibm.db2.jdbc.app.DB2StdXADataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

For more information on DB2 UDB for iSeries, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/

DB2 on z/OS, connecting to Application Server on Windows, UNIX, or workstation-based LINUX

1. **DB2 Universal JDBC Provider**

   The DB2 Universal JDBC Driver is an architecture-neutral JDBC driver for distributed and local DB2 access. Because the Universal Driver architecture is independent of any particular JDBC driver connectivity or target platform, it allows both Java connectivity (Type 4) or Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. Starting with WebSphere Application Server Version 5.0.2, the product now supports both Type 2 and Type 4 JDBC drivers. To use the Type 4 driver, you must install DB2 Version 8.1 or a later version. To use the Type 2 driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version.

   This JDBC driver allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access.

   The DB2 Universal JDBC Driver Provider supports one phase data source:

   com.ibm.db2.jcc.DB2ConnectionPoolDataSource

   Requires JDBC driver files:

   - **db2jcc.jar** After you install DB2, you can find this jar file in the DB2 *java* directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable to point to the **db2jcc.jar** file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** path variable.

Note: To find out the version of the universal driver you are using, issue this DB2 command:

```
java com.ibm.db2.jcc.DB2Jcc -version
```

The output for the above example is:

```
IBM DB2 JDBC Universal Driver Architecture 2.2.xx
```

- **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in a directory defined by **${UNIVERSAL_JDBC_DRIVER_PATH}**environment variable.
- **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  The **db2jcc_license_cisuz.jar** does not ship with Websphere Application Server and should be located in the same directory as the db2jcc.jar file, so that the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** points to both.

```
The classpath for this provider is set as follows:

   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar </classpath>
   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
   <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar</classpath>
```

Note: The license jar files are independent of each other; therefore, order does not matter.

Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
```

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is an actual database name if the **driverType** is set to **4**, or a locally cataloged database name if the **driverType** is set to **2**.
- **driverType** The JDBC connectivity type of a data source. *There are two permitted values: 2 and 4*. If you want to use Universal JDBC Type 2 driver, set this value to 2. If you want to use Universal JDBC Type 4 driver, set this value to 4.
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

2. **DB2 Universal JDBC Provider (XA)**

   The DB2 Universal JDBC Driver (XA) is an architecture-neutral JDBC driver for distributed and local DB2 access. In WebSphere Application Server Version 5.0.2, this driver only supports Java Native Interface (JNI) based connectivity (Type 2) in a single driver instance to DB2. To use this driver, you must install DB2 Version 8.1 Fix Pack 2 or a later version. This driver supports two phase transactions and the WebSphere Application Server Version 5.0 data source. This driver allows applications to use both JDBC and SQLJ access.

   The DB2 Universal JDBC Driver Provider supports the two phase data source:

   com.ibm.db2.jcc.DB2XADataSource

   Requires JDBC driver files:

   - **db2jcc.jar** After you install DB2, you can find this `.jar` file in the DB2 java directory. For Type 4 JDBC driver support from a client machine where DB2 is not installed, copy this file to the local machine. If you install any fixes or upgrades to DB2, you must update this file as well. You must also set the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable to point to the

db2jcc.jar file. See the Cloudscape section for more information on the ***DB2UNIVERSAL_JDBC_DRIVER_PATH*** environment variable.

Note: To find the level of universal driver you are using, issue the following DB2 command:

```
java com.ibm.db2.jcc.DB2Jcc -version
```

example output of the above:

```
IBM DB2 JDBC Universal Driver Architecture 2.2.xx
```

- **db2jcc_license_cu.jar** This is the DB2 Universal JDBC driver license file that allows access to the DB2 Universal database. Use this jar file or the next one to gain access to the database. This jar file ships with WebSphere Application Server in the *WAS_HOME/universalDriver/lib* directory.
- **db2jcc_license_cisuz.jar** This is the DB2 Universal JDBC driver license file that allows access to the following databases:
  - DB2 Universal
  - DB2 for iSeries
  - DB2 for z/OS
  - SQLDS

  You must use the right license jar file to access a specific database backend.

Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
```

Requires a valid authentication alias.

Requires properties:

- **databaseName** This is a locally cataloged database name.
- **driverType** This is the JDBC connectivity type of a data source. *If you are running a version of DB2 prior to DB2 V8.1 FP6, you are restricted to using only the type 2 driver.*
- **serverName** The TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.
- **portNumber** The TCP/IP port number where the DRDA server resides. Provide a value for this property only if your **driverType** is set to **4**. This property is not required if your **driverType** is set to **2**.

3. **DB2 legacy CLI-based Type 2 JDBC Driver**

   Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider instead.

   The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. For use with Application Server on Windows, UNIX, or workstation-based LINUX, this provider requires DB2 Connect (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

   COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

   Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

   Requires **DataStoreHelper** class:

   ```
   com.ibm.websphere.rsadapter.DB2DataStoreHelper
   ```

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider (XA) instead.

The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. For use with Application Server on Windows, UNIX, or workstation-based LINUX, this provider requires DB2 Connect (which is available from DB2).

DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

COM.ibm.db2.jdbc.DB2XADataSource

Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2DataStoreHelper`

Does not require a valid authentication alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

For more information on DB2 for z/OS, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

Cloudscape v10.x

(Cloudscape v10.0 through Cloudscape 10.1.x)

•

Requirement: To configure JDBC access, you must specify Derby JDBC provider names, Derby JDBC driver classes, and Derby data source classes. Because the new Cloudscape code base is the product of the Apache Derby Project, the database uses Derby JDBC classes.

•

Restriction: Do not use Cloudscape v10.x as a production database. Use it for development and test purposes only.

1. **Derby JDBC Provider**

   The Derby JDBC driver provides JDBC access to the Cloudscape v10.x database by using the framework that is already embedded in WebSphere Application Server for Cloudscape. However, you cannot use any Version 4.0 data sources with Cloudscape v10.x.

   The Derby JDBC Provider supports one phase data source:

   org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource

   Requires JDBC driver files: **derby.jar**; full path name: `${WAS_APP_SERVER_ROOT}/derby/lib/derby.jar`

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
     - Example database path name for Windows: `c:\`*`temp\sampleDB`*
     - Example database path name for UNIX or LINUX: `/`*`tmp/sampleDB`*

     If no database currently exists for the path name you want to specify, simply append `;create=true` to the path name to create a database dynamically. (For example: `c:\`*`temp\sampleDB`*`;create=true`)

2. **Derby JDBC Provider (XA)**

   The Derby JDBC driver (XA) provides JDBC access to the Cloudscape v10.x database by using the framework that is already embedded in WebSphere Application Server for Cloudscape. However, you cannot use any Version 4.0 data sources with Cloudscape v10.x.

   The Derby JDBC Provider (XA) supports two phase data source:

   org.apache.derby.jdbc.EmbeddedXADataSource

   Requires JDBC driver files: **derby.jar**; full path name: `${WAS_APP_SERVER_ROOT}/derby/lib/derby.jar`

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
     - Example database path name for Windows: `c:\temp\sampleDB`
     - Example database path name for UNIX or LINUX: `/tmp/sampleDB`

     If no database currently exists for the path name you want to specify, simply append `;create=true` to the path name to create a database dynamically. (For example: `c:\temp\sampleDB;create=true`)

3. **Derby Network Server using Universal JDBC driver** -- Deprecated

   This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console.

   This Derby driver takes advantage of the Network Server support that the DB2 universal Type 4 JDBC driver provides. You cannot use any Version 4.0 data sources with Cloudscape v10.x.

   Use the following one phase data source for the Derby Network Server using the Universal JDBC driver:

   com.ibm.db2.jcc.DB2ConnectionPoolDataSource

   Requires JDBC driver files:

   - **db2jcc.jar** If you install and run DB2, you must use the **db2jcc.jar** file that comes with DB2. To do that, the classpath in the JDBC template for Derby Network Server is set to be:

     ```
     <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar</classpath>

     <classpath>${CLOUDSCAPE_JDBC_DRIVER_PATH}/otherJars/db2jcc.jar</classpath>

     <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
     ```

     which means that the db2jcc.jar from DB2 always takes precedence. Note that this also means that you must set the DB2 environment variable **DB2UNIVERSAL_JDBC_DRIVER_PATH** in WebSphere Application Server when you set up your DB2 data source. This is instead of hard coding the path of the db2jcc.jar for DB2 data sources.
   - **db2jcc_license_cu.jar** This file is the DB2 Universal JDBC license file that provides access to the Derby databases using the **Network Server** framework. Use this file to gain access to the database. This file ships with WebSphere and is located in **${UNIVERSAL_JDBC_DRIVER_PATH}**.

     Note: **UNIVERSAL_JDBC_DRIVER_PATH** is a WebSphere environment variable that is already mapped to the location in Websphere Application Server where the license jar file is located, and will only be used if the **DB2UNIVERSAL_JDBC_DRIVER_PATH** is not set. DB2 users should ensure that **DB2UNIVERSAL_JDBC_DRIVER_PATH** is set to avoid loading multiple versions of the db2jcc.jar file.

     Note: **DB2UNIVERSAL_JDBC_DRIVER_PATH** is a WebSphere environment variable that you must set to point to the location of db2jcc.jar file (that comes with DB2). This variable is set only if you create a DB2 provider.

Note: Derby requires only db2jcc_license_c.jar; however, WebSphere Application Server uses db2jcc_license_cu.jar because this works for both DB2 UDB and Derby.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
  - Example database path name for Windows: `c:\temp\sampleDB`
  - Example database path name for UNIX or LINUX: `/tmp/sampleDB`

  If no database currently exists for the path name you want to specify, simply append `;create=true` to the path name to create a database dynamically. (For example: `c:\temp\sampleDB;create=true`)
- **driverType** Only the Type 4 driver is allowed.
- **serverName** The TCP/IP address or the host name for the Distributed Relational Database Architecture (DRDA) server.
- **portNumber** The TCP/IP port number where the DRDA server resides. The default value is port ***1527***.
- **retrieveMessagesfromServerOnGetMessage** This property is required by WebSphere Application Server, not the database. The default value is ***false***. You must set the value of this property to ***true***, to enable text retrieval using the **SQLException.getMessage()** method.

4. **Derby Network Server using Derby Client provider**

   Use this provider to access only Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server.

   Use the following one phase data source for the Derby Network Server using Derby Client provider:

   org.apache.derby.jdbc.ClientConnectionPoolDataSource

   Cloudscape v10.1.x does not support Version 4.0 data sources.

   Requires the following JDBC driver file:
   - **derbyclient.jar**
   -

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

   Requires the **databaseName** property: The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
   - Example database path name for Windows: `c:\temp\sampleDB`
   - Example database path name for UNIX or LINUX: `/tmp/sampleDB`

   If no database currently exists for the path name you want to specify, append `;create=true` to the path name to create a database dynamically. (For example: `c:\temp\sampleDB;create=true`)

5. **Derby Network Server using Derby Client provider (XA)**

   Use this provider to access only Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server.

   Use the following XA data source for this Derby Network Server using Derby Client provider:

   org.apache.derby.jdbc.XADataSource

   Cloudscape v10.1.x does not support Version 4.0 data sources.

   Requires the following JDBC driver file:
   - **derbyclient.jar**
   -

   Requires **DataStoreHelper** class:

com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper

Requires the **databaseName** property: The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of WAS_HOME/derby (or the equivalent default for a UNIX or LINUX environment).
- Example database path name for Windows: c:*temp**sampleDB*
- Example database path name for UNIX or LINUX: /*tmp*/*sampleDB*

If no database currently exists for the path name you want to specify, append ;create=true to the path name to create a database dynamically. (For example: c:*temp**sampleDB*;create=true)

For more information on the new Cloudscape code base, visit the Apache Derby Project Web site at: http://incubator.apache.org/derby/.

Informix

1. **Informix JDBC Driver**

   The Informix JDBC Driver is a Type 4 JDBC driver that provides JDBC access to the Informix database.

   Informix JDBC Driver supports one phase data source:

   com.informix.jdbcx.IfxConnectionPoolDataSource

   Requires JDBC driver files:

   ifxjdbc.jar
   ifxjdbcx.jar

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.InformixDataStoreHelper

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the Informix instance on the server. Example: *ol_myserver*.
   - **portNumber** The port on which the instances listen. Example: *1526*.
   - **ifxIFXHOST** Either the IP address or the host name of the machine that is running the Informix database to which you want to connect. Example: *myserver.mydomain.com*.

     To support IPv6: On AIX and Solaris, IBM Informix Dynamic Server 10.00 with fix pack 1 supports the IPv6 standard. To enable IPv6 on your WebSphere Application Server connection with one of these Informix releases, input your full IPv6 host name for the ifxIFXHOST property.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **informixLockModeWait** Although not required, this property enables you to set the number of seconds that Informix software waits for a lock. By default, Informix code throws an exception if it cannot immediately acquire a lock. Example: *2*.

2. **Informix JDBC Driver (XA)**

   The Informix JDBC Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Informix database.

   Informix JDBC Driver (XA) supports two phase data source:

   com.informix.jdbcx.IfxXADataSource

   Requires JDBC driver files:

   ifxjdbc.jar
   ifxjdbcx.jar

   To use SQLJ: This provider also requires driver file **ifxsqlj.jar** if you plan to use SQLJ for queries.

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.InformixDataStoreHelper

   Requires a valid authentication alias.

Requires properties:
- **serverName** The name of the Informix instance on the server. Example: *ol_myserver*.
- **portNumber** The port on which the instances listen. Example: *1526*.
- **ifxIFXHOST** Either the IP address or the host name of the machine that is running the Informix database to which you want to connect. Example: *myserver.mydomain.com*.

  To support IPv6: On AIX and Solaris, IBM Informix Dynamic Server 10.00 with fix pack 1 supports the IPv6 standard. To enable IPv6 on your WebSphere Application Server connection with one of these Informix releases, input your full IPv6 host name for the ifxIFXHOST property.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
- **ifxIFX_XASPEC** Turn on this property when multiple users access the same database. Activating the property enforces tight coupling of XA transactions within the same global transaction ID, and requires the transactions to share lock space. These parameters help prevent transaction management errors from occurring in cases of multiple client requests. Turn on the `ifxIFX_XASPEC` property by assigning it the value of `Y` or `y`; either character works because the setting is not case-specific. Turn the property off by assigning it the value of `N` or `n`. WebSphere Application Server ignores all other values. Your setting for the property overrides the Informix database system setting.
- **informixLockModeWait** Although not required, this property enables you to set the number of seconds that Informix software waits for a lock. By default, Informix code throws an exception if it cannot immediately acquire a lock. Example: *2*.

For more information on Informix, visit the Informix Web site at: http://www.ibm.com/software/data/informix/

Sybase

1. **Sybase jConnect for JDBC driver**

   The Sybase jConnect JDBC driver is a Type 4 JDBC driver that provides JDBC access to the Sybase database.

   Sybase jConnect JDBC driver supports one phase data source:

   `com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource`

   Requires JDBC driver files: **`jconn2.jar`**.

   For IPv6 support: For applications that access Sybase in an IPv6 environment, you must use the Sybase jConnect JDBC driver version 6.0 EBF 12884. This implementation uses different classes and therefore requires a different JAR file and a different implementation class designation. You can define the jConnect JDBC driver v6.0 EBF 12884 in the administrative console by selecting **User-defined** as the database type on the New JDBC provider page. On the JDBC provider general configuration page, replace the default JAR file name with **`jconn3.jar`**. For the implementation class, input **`com.sybase.jdbc3.jdbc.SybConnectionPoolDataSource`**. The data store helper class and required properties are the same for all Sybase JDBC drivers.

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the database server. Example: *myserver.mydomain.com*.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.
   - **connectionProperties** A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true`(Type: java.lang.String)

2. **Sybase jConnect for JDBC driver (XA)**

   The Sybase jConnect JDBC driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Sybase database.

   Sybase jConnect JDBC driver (XA) supports two phase data source:

   com.sybase.jdbc2.jdbc.SybXADataSource

   Requires JDBC driver files: **jconn2.jar**.

   For IPv6 support: For applications that access Sybase in an IPv6 environment, you must use the Sybase jConnect JDBC driver version 6.0 EBF 12884. This implementation uses different classes and therefore requires a different JAR file and a different implementation class designation. You can define the jConnect JDBC driver v6.0 EBF 12884 in the administrative console by selecting **User-defined** as the database type on the New JDBC provider page. On the JDBC provider general configuration page, replace the default JAR file name with **jconn3.jar**. For the implementation class, input **com.sybase.jdbc3.jdbc.SybXADataSource**. The data store helper class and required properties are the same for all Sybase JDBC drivers.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the database server. Example: *myserver.mydomain.com*
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.
   - **connectionProperties** A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true`(Type: java.lang.String)

For more information on Sybase, visit the Sybase Web site at: http://www.sybase.com/

Oracle

1. **Oracle JDBC Driver**

   The Oracle JDBC Driver provides JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

   Oracle JDBC Driver supports one phase data source:

   oracle.jdbc.pool.OracleConnectionPoolDataSource

   Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   (Note: If you are running Oracle10g, use `com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper`.)

   Requires a valid authentication alias.

   Requires properties:
   - **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@myServer:1521:myDatabase*, where *myServer* is the server name, *1521* is the port it is using for communication, and *myDatabase* is the database name.

2. **Oracle JDBC Driver (XA)**

   The Oracle JDBC Driver (XA) provides XA-compliant JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

   Oracle JDBC Driver (XA) supports two phase data source:

   oracle.jdbc.xa.client.OracleXADataSource

Requires JDBC driver files: **ojdbc14.jar**. (Note: If you require Oracle trace, use **ojdbc14_g.jar**.)

Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.OracleDataStoreHelper
```

(Note: If you are running Oracle10g, use `com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper`.)

Requires a valid authentication alias.

Requires properties:

- **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@myServer:1521:myDatabase*, where *myServer* is the server name, *1521* is the port it is using for communication, and *myDatabase* is the database name.

For more information on Oracle, visit the Oracle Web site at: http://www.oracle.com/

MS SQL Server

MS SQL Server 2005: WebSphere Application Server Version 6.10 supports JDBC transactions with the new Microsoft SQL Server 2005, as well as Microsoft SQL Server 2000. For access to either version of the database, use Version 3.5, Service pack 2, of either of the following JDBC providers:

- DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server -- including the XA implementation
- IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server -- including the XA implementation

These JDBC drivers provide the same function for MS SQL Server 2005 as MS SQL Server 2000. As long as you use only those new features of MS SQL Server 2005 that have no impact on JDBC transactions, you generally risk no exceptions by upgrading to the new version. WebSphere Application Server does, however, support two new options for setting isolation level in MS SQL Server 2005: SNAPSHOT and READ_COMMITTED_SNAPSHOT. The following chart describes these isolation levels as well as the few requirements and concerns for using MS SQL Server 2005 with Application Server v6.1.

Compatibility or New Usage concern		Resolution or Recommended action
New requirements:	Specifying your database version	You do not have to specify your version of MS SQL Server. Specify the name of your database as usual. WebSphere Application Server detects the version and makes any necessary class attribute adjustments.
	Parenthesis requirement for locking hints	MS SQL Server 2005 requires that you place parentheses around locking hints. For example: `select value from t1 (holdlock)` `where name = ?`
	New syntax for joining multiple locking hints	• MS SQL Server 2005 requires use of the keyword **with** to join multiple locking hints. For example: `select value from t1 with (updlock rowlock)` `where name = ?` • For MS SQL Server 2000, no keyword is required. For example: `select value from t1 (updlock rowlock)` `where name = ?`
	Use of the `alternate servers` custom data source property	Verify that all application component clients of the data source issue commands that are valid for both database versions before you configure the `alternate servers` property to include both MS SQL Server 2005 and Server 2000 machines.
	Deprecated data types	Microsoft deprecated three data types for SQL Server 2005, which are shown in the following list along with each replacement data type: • `text`, replaced by `varchar(max)` • `ntext`, replaced by `nvarchar(max)` • `image`, replaced by `varbinary(max)`

Compatibility or New Usage concern		Resolution or Recommended action
New features:	SNAPSHOT isolation level	This new isolation level implements optimistic locking for transactions in which MS SQL Server 2005 serializes the data. You must configure the ALLOW_SNAPSHOT_ISOLATION setting on the database, and then set the isolation level in one of two ways: • By isolation level constant: Invoke the method **setTransactionIsolation** with one of three new attributes: – `conn.setTransactionIsolation (com.ibm.websphere.jdbc.extensions. ExtConstants.TRANSACTION_SNAPSHOT)` – `conn.setTransactionIsolation (com.ddtek.jdbc.extensions. ExtConstants.TRANSACTION_SNAPSHOT)` – `conn.setTransactionIsolation(16)` • By custom data source property: – Set the new data source custom property **snapshotSerializable** to true, ***and*** – Invoke the method **setTransactionIsolation** with the attribute: `conn.setTransactionIsolation (java.sql.Connection.TRANSACTION_SERIALIZABLE)`
	READ_COMMITTED_SNAPSHOT isolation level	This isolation level is a new implementation of Read committed. The policy enforces optimistic locking for read operations with MS SQL Server 2005. You must configure the isolation level on the database. Then invoke the method **setTransactionIsolation** with the attribute: `conn.setTransactionIsolation (java.sql.Connection.TRANSACTION_READ_COMMITTED)`
	Transact-SQL enhancements, including: new functions, additional data types, and the ability to create recursive queries	WebSphere Application Server does not currently support these features; do not use them with WebSphere Application Server Version 6.10.

Consult the Microsoft Web page at http://msdn2.microsoft.com/en-us/library/ms143232(en-US.SQL.90).aspx for a complete list of deprecated items, as well as backward compatibility provisions, for MS SQL Server 2005.

For the MS SQL Server JDBC drivers that support both database versions, perform the same steps and set the same class paths and properties that were previously required for MS SQL Server 2000.

1. **DataDirect ConnectJDBC type 4 driver for MS SQL Server**

   DataDirect ConnectJDBC type 4 driver for MS SQL Server is a Type 4 JDBC driver that provides JDBC access to the MS SQL Server 2005 and MS SQL Server 2000 databases. This provider is for use only with the Connect JDBC driver purchased from DataDirect Technologies.

   This JDBC provider supports this data source:

   com.ddtek.jdbcx.sqlserver.SQLServerDataSource

   Requires JDBC driver files:

   sqlserver.jar,
   base.jar and **util.jar**

(The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file is not in the same directory as the other three jar files. Instead, it is located in the **../spy/** directory.)

Requires **DataStoreHelper** class:

com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

2. **DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA)**

   DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA) is a Type 4 JDBC driver which provides XA-compliant JDBC access to the MS SQL Server 2005 and MS SQL Server 2000 databases. This provider is for use only with the Connect JDBC driver purchased from DataDirect Technologies.

   This JDBC provider supports this data source:

   com.ddtek.jdbcx.sqlserver.SQLServerDataSource.

   Requires JDBC driver files:

   sqlserver.jar,
   base.jar and **util.jar.**

   (The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file is not in the same directory as the other three jar files. Instead, it is located in the **../spy/** directory.)

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

   For more information on the DataDirect ConnectJDBC driver, visit the DataDirect Web site at: http://www.datadirect-technologies.com/

3. **IBM WebSphere embedded ConnectJDBC driver for MS SQL Server**

   WebSphere embedded ConnectJDBC driver for MS SQL Server is a Type 4 JDBC driver that provides JDBC access to the MS SQL Server 2005 and MS SQL Server 2000 databases. This JDBC driver ships with WebSphere Application Server. Only use this provider with the Connect JDBC driver embedded in WebSphere; it cannot be used with a Connect JDBC driver purchased separately from DataDirect Technologies.

   This JDBC provider supports this data source:

   com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.

   Requires JDBC driver files:

   sqlserver.jar
   base.jar and
   util.jar.

(The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file for the WebSphere embedded Connect JDBC driver ships with WebSphere Application Server. All the files are located in the `WAS_HOME/lib/` directory.)

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **IBM WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)**

   WebSphere embedded ConnectJDBC driver for MS SQL Server (XA) is a Type 4 JDBC driver that supports two-phase commit transactions on connections with the MS SQL Server 2005 and MS SQL Server 2000 databases. This JDBC driver ships with WebSphere Application Server. Use this provider with the IBM WebSphere Connect JDBC driver embedded in WebSphere Application Server. Do not use it with the DataDirect Connect JDBC driver purchased separately from DataDirect Technologies.

   The ConnectJDBC provider supports the following data source:

   com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.

   Requires JDBC driver files:

   sqlserver.jar
   base.jar and
   util.jar.

   An additional file, the **spy.jar** file, is optional. You need **spy.jar** for spy logging, which is a form of JDBC driver-level trace.

   All of the JAR files in the previous list are shipped with WebSphere Application Server and are installed automatically with the product. They are also updated automatically when you apply WebSphere Application Server service packs.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

   Patches to the IBM WebSphere Connect JDBC driver jar files are installed automatically when you apply WebSphere Application Server service packs. However, to update Microsoft SQL Server-side programs for this JDBC driver, you must go to the IBM FTP site for WebSphere Application Server embedded product updates. Use the following URL:

   ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm

   An important server-side program is Stored Procedures for the Java Transaction API (JTA). Whether you need to run one or two phase transactions with the XA-enabled IBM WebSphere Connect JDBC driver, you must install Stored Procedures for JTA on all machines that run Microsoft SQL. The WebSphere Application Server installation disks contain a base level of Stored Procedures for JTA. Go to the previously listed FTP site for updates to this API.

   Install Stored Procedures for JTA by performing the following steps:

a. Determine whether you are running the 32-bit or 64-bit MS SQL Server and select the appropriate `sqljbc.dll` and `instjdbc.sql` files.
b. Stop your MS SQL Server service.
c. Copy the `sqljdbc.dll` file into your `%SQL_SERVER_INSTALL%\Binn\` directory.
d. Restart the MS SQL Server service.
e. Run the `instjdbc.sql` script. (The script can be run by the MS SQL Server Query Analyzer or the ISQL utility).

`ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm`

5. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server** -- Deprecated

   This type 3 JDBC driver for MS SQL Server is deprecated in WebSphere Application Server Version 6.0. Therefore it is no longer an available choice among provider types in the administrative console.

   For best results with WebSphere Application Server JDBC access to MS SQL Server, use only JDBC drivers that are *not* marked for deprecation. However, if you must continue using a deprecated driver for JDBC access to MS SQL Server, you can configure it through the WebSphere Application Server administrative console. Be sure to select **User-defined** for the database type. This selection triggers the console to display default class files, data source interfaces, and so on for your user-defined JDBC provider type. Replace those defaults with the following settings that are specific to the DataDirect SequeLink type 3 JDBC driver. For information on this task, read the Configuring a JDBC provider using the administrative console section of the *Administering applications and their environment* PDF book.

   Incompatible with MS SQL Server 2005: Use this JDBC driver for access to MS SQL Server 2000 only.

   DataDirect SequeLink type 3 JDBC driver supports the following data source:

   `com.ddtek.jdbcx.sequelink.SequeLinkDataSource`

   Requires JDBC driver files:

   `sljc.jar` and
   `spy-sl.jar`

   (The JDBC driver shipped with WebSphere Application Server requires the **`sljc.jar`** and the **`spy-sl.jar`** files. The JDBC driver purchased from DataDirect requires the **`sljc.jar`** and the **`spy.jar`** files. The **`spy.jar`** and **`spy-sl.jar`** files are optional. You need these files to enable spy logging.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.SequeLinkDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the server in which SequeLink Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that SequeLink Server uses for communication. By default, SequeLink Server uses port 19996.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **enable2phase** This property is necessary only if your application must participate in two-phase transactions. By default, Application Server sets a user-defined implementation type to a connection pool data source. The connection pool data source supports only one-phase transactions. Configure two-phase transaction support by setting the *enable2Phase* custom property on each data source that you create with your user-defined JDBC provider. Follow these steps:
     a. Go to the Custom properties administrative console page by clicking **Resources** > **JDBC Providers** > *your_JDBC_provider* > **Data sources** > *your_data_source* > **Custom properties**.
     b. Click **New**.
     c. Input `enable2Phase` as the property name and assign it the value of `true`. For Version 4 data sources, use a property that works in the opposite manner: Input `disable2Phase` and assign it the value of `false`.

The DataDirect SequeLink type 3 JDBC driver requires installation of SequeLink Server on all machines running MS SQL Server. See the readme.html file found in the DataDirect folder on the WebSphere Application Server CD for instructions on how to install SequeLink Server. (Install SequeLink Server from the WebSphere Application Server CD only if you are using the SequeLink JDBC driver embedded in WebSphere. Otherwise, install a copy of SequeLink Server purchased from DataDirect Technologies.)

Patches to the IBM WebSphere SequeLink JDBC driver jar files are installed automatically when applying WebSphere Application Server service packs. If updates are ever needed for the Microsoft SQL Server-side installables (SequeLink server) for the IBM WebSphere SequeLink JDBC driver, they will be made available from the following FTP site:

`ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm`

For more information on the DataDirect SequeLink type 3 JDBC driver, visit the DataDirect Web site at:

`http://www.datadirect-technologies.com/`

6. **Microsoft JDBC driver for MS SQL Server 2000** -- Deprecated

   This type 4 JDBC driver for MS SQL Server 2000 is deprecated in WebSphere Application Server Version 6.0. Therefore it is no longer an available choice among provider types in the administrative console.

   For best results with WebSphere Application Server JDBC access to MS SQL Server, use only JDBC drivers that are *not* marked for deprecation. However, if you must continue using a deprecated driver for JDBC access to MS SQL Server, you can configure it through the WebSphere Application Server administrative console. Be sure to select **User-defined** for the database type. This selection triggers the console to display default class files, data source interfaces, and so on for your user-defined JDBC provider type. Replace those defaults with the following settings that are specific to the Microsoft JDBC driver for MS SQL Server 2000. Follow the steps listed in Configuring a JDBC provider using the administrative console section of the *Administering applications and their environment* PDF book..

   Microsoft JDBC driver for MS SQL Server 2000 supports the following data source:

   `com.microsoft.jdbcx.sqlserver.SQLServerDataSource`

   Requires JDBC driver files:

   `mssqlserver.jar`,
   `msbase.jar` and **`msutil.jar`**

   (The **`spy.jar`** file is optional. You need it to enable spy logging. However, Microsoft does not ship the **`spy.jar`** file. Contact Microsoft about this issue.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **enable2phase** This property is necessary only if your application must participate in two-phase transactions. By default, Application Server sets a user-defined implementation type to a connection pool data source. The connection pool data source supports only one-phase transactions. Configure two-phase transaction support by setting the *enable2Phase* custom property on each data source that you create with your user-defined JDBC provider. Follow these steps:
     a. Go to the Custom properties administrative console page by clicking **Resources** > **JDBC Providers** > *your_JDBC_provider* > **Data sources** > *your_data_source* > **Custom properties**.
     b. Click **New**.
     c. Input `enable2Phase` as the property name and assign it the value of `true`. For Version 4 data sources, use a property that works in the opposite manner: Input `disable2Phase` and assign it the value of `false`.

For more information on the Microsoft JDBC driver for MS SQL Server 2000, visit the Microsoft Web site at:

`http://www.microsoft.com/sql`

Configuring a JDBC provider using the administrative console

To create connections between an application and a relational database, WebSphere Application Server uses the driver implementation classes that are encapsulated by the JDBC provider.

Each JDBC provider is essentially an object that represents vendor-specific JDBC driver classes to Application Server, for establishing access to that particular vendor database. JDBC providers are prerequisites for data sources, which supply applications with the physical connections to a database. Consult the JDBC provider table to identify the appropriate JDBC provider for your database and application requirements.

Configure at least one JDBC provider for each database server that you plan to use at a particular scope within your WebSphere Application Server environment.

1. Open the administrative console.
2. Click **Resources** > **JDBC** > **JDBC Providers**.
3. Select the *scope* at which applications can use the JDBC provider. (This scope becomes the scope of any data source that you associate with this provider.) You can choose a cell, node, cluster, or server. For more information, see Administrative console scope settings.
4. Click **New**. This action causes the **Create a new JDBC Provider** wizard to launch.
5. Use the first drop-down list to select the database type of the JDBC provider that you need to create.

   The User-Defined option: Select **User-Defined** for your database type if you encounter either of the following scenarios:

   - You do not see your database type.
   - You cannot select the JDBC provider type that you need in the next step.

   The user-defined selection triggers the wizard page to display your provider type as a User-defined JDBC provider, and your implementation type as User-defined. Consult your database documentation for the JDBC driver class files, data source properties, and so on that are required for your user-defined provider. You must supply this information on the next two wizard pages: one page for database class path information, and the other page for database-specific properties.
6. Select your JDBC provider type if it is displayed in the second drop-down list. Select **Show Deprecated** to trigger the display of both current and deprecated providers. If you cannot find your provider in this expanded list, then select **User-Defined** from the previous list of database types.
7. From the third drop-down list, select the implementation type that is necessary for your application. If your application does not require that connections support two-phase commit transactions, choose **Connection Pool Data Source**. Choose **XA Data Source**, however, if your application requires connections that support two-phase commit transactions. Applications that use this data source configuration have the benefit of container-managed transaction recovery.

   After you select an implementation type, the wizard fills the name and the description fields for your JDBC provider. You can type different values for these fields; they exist for administrative purposes only.
8. Click **Next** to see the **Enter database class path information** wizard page.
9. In the Class path field, type the full path location of the database JDBC driver class files. Your class path information becomes the value of the WebSphere environment variable that is displayed on this page, in the form of ${*DATABASE*_JDBC_DRIVER_PATH}. WebSphere Application Server uses the variable to define your JDBC provider; this practice eliminates the need to specify static JDBC class

paths for individual applications. Remember that if you do not provide the full, correct JDBC driver class path for the variable, your data source ultimately fails. If the field already displays a fully qualified class path, you can accept that variable definition by completing the rest of this wizard page and clicking **Next**.

10. Use the Native library path field to specify additional class files that your JDBC driver might require to function properly on your WebSphere Application Server platform. Type the full directory path name of these class files.
11. Click **Next** to see a summary of your JDBC provider settings.
12. Click **Finish** if you are satisfied with the entire JDBC provider configuration. You now see the JDBC provider collection page, which displays your new JDBC provider in a table along with other providers that are configured for the same scope.

Your next step is to create a data source for association with your JDBC provider. For detailed information, see “Configuring a data source using the administrative console” on page 721.

JDBC provider collection:

Use this page to view JDBC providers. The JDBC provider object encapsulates the specific JDBC driver implementation class for the data sources that you define and associate with the provider.

To view this administrative console page, click **Resources** > **JDBC** > **JDBC providers** .

Name:

Specifies a text identifier for this provider.

For example, this field can be *DB2 JDBC Provider (XA)*.

Data type String

Scope:

Specifies the scope of the JDBC provider; if you use any scope other than the default of *Node*, the provider might not be available in other scope contexts. Data sources that are created with this JDBC provider inherit this scope.

Description:

Specifies a text string describing this provider.

Data type String

JDBC provider settings:

Use this page to modify JDBC provider settings.

To view this administrative console page, click **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider*.

Important: If you use this page to modify the class path or native library path of an existing JDBC provider: After you apply and save the new settings, you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

Scope:

Specifies the scope of the JDBC provider; data sources that are created with this JDBC provider inherit this scope.

Name:

Specifies the name of the resource provider.

Data type String

Description:

Specifies a text description for the resource provider.

Data type String

Class path:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

For example, *D:/SQLLIB/java/db2java.zip*.

Class path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Class paths contain variable (symbolic) names which you can substitute using a variable map. Check the driver installation notes for the specific required JAR file names.

Data type String

Native Library Path:

Specifies a list of paths that forms the location for the resource provider native libraries.

Native path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Native paths can contain variable (symbolic) names which you can substitute using a variable map.

Data type String

Implementation class name:

Specifies the Java class name of the JDBC driver implementation.

This class is available in the driver file mentioned in the class path description above. For example, *COM.ibm.db2.jdbc.DB2XADataSource*.

Note: If you modify the implementation class name of the JDBC provider after you have created the provider, you might disconnect the provider from the template used to create it. As a result, data sources created from this JDBC provider do not have an associated template; you must manually configure a working data source through setting custom properties.

Data type String

JDBC provider summary:

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 on Windows, UNIX, or workstation-based LINUX	DB2 Universal JDBC Provider	One phase only	
	DB2 Universal JDBC Provider (XA)	One and two phase	The XA implementation is *not* supported in WebSphere Application Server run on workstation-based LINUX
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 UDB for iSeries	DB2 UDB for iSeries (Native)	One phase only	Recommended for use with WebSphere Application Server **run on iSeries**
	DB2 UDB for iSeries (Native XA)	One and two phase	Recommended for use with WebSphere Application Server **run on iSeries**
	DB2 UDB for iSeries (Toolbox)	One phase only	
	DB2 UDB for iSeries (Toolbox XA)	One and two phase	
	DB2 Universal JDBC Provider (XA)	One and two phase	**-***Only* for use with WebSphere Application Server **run on z/OS** **-**Only driver type 4 is supported **-**Does *not* support Version 4 data sources
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect driver (available from DB2)
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect driver (available from DB2)

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 on z/OS	DB2 for z/OS Local JDBC Provider (RRS), using the Legacy DB2 for OS/390 and z/OS JDBC Driver **Removed support:** WebSphere Application Server v6.1 does not support this provider. Use the DB2 Universal JDBC Driver provider instead. See ″Migrating from the JDBC/SQLJ Driver for OS/390 and z/OS to the DB2 Universal JDBC Driver″ in the Information Management Software for z/OS Solutions Information Center.	One and two phase	*Only* for use with WebSphere Application Server **run on z/OS**
	DB2 Universal JDBC Provider	One phase only	
	DB2 Universal JDBC Provider (XA)	One and two phase	**-**Only driver type 4 is supported in WebSphere Application Server **run on z/OS** **-**Does *not* support Version 4 data sources in WebSphere Application Server **run on z/OS**
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect program (available from DB2)
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect program (available from DB2)

Database type	JDBC Provider	Transaction support	Version and other considerations
Cloudscape Version 10.1.x *Cloudscape v10.1.x provides the Apache Derby runtime.* **JDBC naming requirements:** Because the new Cloudscape code base is the product of the Apache Derby Project, Cloudscape v10.1.x uses Derby JDBC classes. Therefore you must specify Derby JDBC provider names, Derby JDBC driver classes, and Derby data source classes to configure JDBC access to Cloudscape v10.1.x.	Derby JDBC Provider	One phase only	- Not for use in clustered environment: accessible from a single JVM only - Does not support Version 4 data sources
	Derby JDBC Provider (XA)	One and two phase	- Not for use in clustered environment: accessible from a single JVM only - Does not support Version 4 data sources
	Derby Network Server Provider using the Universal JDBC driver *Deprecated in WebSphere Application Server v.6.1*	One phase only	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support XA - Does not support Version 4 data sources
	Derby Network Server using Derby Client provider	One phase only	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support Version 4 data sources - Only for use with Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server
	Derby Network Server using Derby Client provider (XA)	One and two phase	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support Version 4 data sources - Only for use with Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server
Informix	Informix JDBC Provider	One phase only	
	Informix JDBC Provider (XA)	One and two phase	
Sybase	Sybase jConnect for JDBC Provider	One phase only	
	Sybase jConnect for JDBC Provider (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
Oracle	Oracle JDBC Provider	One phase only	
	Oracle JDBC Provider(XA)	One and two phase	
Microsoft SQL Server	DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server	One phase only	- Only for use with the corresponding driver from DataDirect Technologies - Version 3.5, Service pack 2 of the DataDirect ConnectJDBC type 4 driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server (XA)	One and two phase	- Only for use with the corresponding driver from DataDirect Technologies - Version 3.5, Service pack 2 of the DataDirect ConnectJDBC type 4 driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server	One phase only	- Cannot be used outside of WebSphere Application Server environment - Version 3.5, Service pack 2 of the driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server (XA)	One and two phase	- Cannot be used outside of WebSphere Application Server environment - Version 3.5, Service pack 2 of the driver can support access to both MS SQL Server 2005 and MS SQL Server 2000

Configuring a data source using the administrative console

Application components use a data source to access connection instances to a relational database.

WebSphere Application Server supports two different versions of data source. Determine the data source for your environment according to the enterprise bean and servlet specification levels that are the basis of your applications:

- *Data sources (WebSphere Application Server Version 4)* are for use with the Enterprise JavaBeans (EJB) 1.0 specification and the Java Servlet 2.2 specification.
- Data sources of the latest standard version are for use with applications that implement the more advanced releases of these specifications.

Potential deprecation issue: This task gives you the option to create a new JDBC provider for the data source. Consult the article “Vendor-specific data sources minimum required

settings" on page 687 to see which JDBC providers are deprecated. The following procedure includes steps for designating a deprecated provider if necessary.

When you create a data source, you associate it with a JDBC provider that is configured for access to a specific vendor database. WebSphere Application Server requires both objects for your applications to make calls to that particular database and receive data from it. The data source provides connection management capabilities that physically make possible these exchanges between your applications and your databases.

1. Open the administrative console.
2. You can access the necessary console page either by clicking **Resources** > **JDBC** > **Data sources**, or by clicking **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources**.
3. Select the *scope* at which applications can use the data source. You can choose a cell, node, cluster, or server. For more information, see the Administrative console scope settings article.
4. Click **New**. This action causes the **Create a data source** wizard to launch and display the Enter basic data source information page. The first field is the scope field, which is read-only. This field displays your previous scope selection.
5. Type a data source name in the Data source name field. This name identifies the data source for administrative purposes only.
6. Type a Java Naming and Directory Interface (JNDI) name in the JNDI name field. WebSphere Application Server uses the JNDI name to bind application resource references to this data source. For more information on JNDI, consult the "Naming" on page 1772 article.
7. Set a component-managed alias to secure your data source. A component-managed alias represents a combination of ID and password that is specified in an application for data source authentication. Therefore, the alias that you set on the data source must be identical to the alias in the application code. For more information on Java 2 Connector (J2C) security, see the Managing J2EE Connector Architecture authentication data entries article.

   Configuring an alias is an optional step. If you do not require application components to authenticate with the data source, select *(none)* from the drop-down list.

   To set a component-managed alias, either select an existing alias or create a new one.

   - Use the drop-down list to select an existing component-managed authentication alias.
   - To create a new alias, click the **create a new one** link. This action closes the data source wizard and triggers the administrative console to display the J2C authentication data collection page. Click **New** to define a new alias. Click **OK** to save your settings and view the new alias on the J2C authentication data collection page. Restart the data source wizard by navigating back to the data source collection page, selecting the appropriate scope, and clicking **New**.
8. Click **Next** to see the wizard page Select JDBC provider.
9. Either select an existing JDBC provider, or create a new provider.

   To select an existing JDBC provider:

   a. Click **Select an existing JDBC provider**.

   b. Select a JDBC driver from the drop-down list.

   c. Click **Next**. You now see the page entitled Enter database specific properties for the data source.

   To create a new JDBC provider:

   a. Click **Create new JDBC provider**.

   b. Click **Next** to see the Create JDBC provider page.

   c. Use the first drop-down list to select the database type of the JDBC provider that you need to create.

      The User-Defined option: Select **User-Defined** for your database type if you encounter either of the following scenarios:

- You do not see your database type.
- You cannot select the JDBC provider type that you need in the next step.

The user-defined selection triggers the wizard page to display your provider type as a User-defined JDBC provider, and your implementation type as User-defined. Consult your database documentation for the JDBC driver class files, data source properties, and so on that are required for your user-defined provider. You must supply this information on the next two wizard pages: one page for database class path information, and the other page for database-specific properties.

d. Select your JDBC provider type if it is displayed in the second drop-down list. Select **Show Deprecated** to trigger the display of both current and deprecated providers. If you cannot find your provider in this expanded list, then select **User-Defined** from the previous list of database types.

e. From the third drop-down list, select the implementation type that is necessary for your application. If your application does not require that connections support two-phase commit transactions, choose **Connection Pool Data Source**. Choose **XA Data Source**, however, if your application requires connections that support two-phase commit transactions. Applications that use this data source configuration have the benefit of container-managed transaction recovery.

After you select an implementation type, the wizard fills the name and the description fields for your JDBC provider. You can type different values for these fields; they exist for administrative purposes only.

f. Click **Next** after you have defined your database type, provider type, and implementation type. Now you see the wizard page Enter database class path information.

g. In the class path field, type the full path location of the database JDBC driver class files. Your class path information becomes the value of the WebSphere environment variable that is displayed on this page, in the form of ${*DATABASE*_JDBC_DRIVER_PATH}. WebSphere Application Server uses the variable to define your JDBC provider; this practice eliminates the need to specify static JDBC class paths for individual applications. Remember that if you do not provide the full, correct JDBC driver class path for the variable, your data source ultimately fails. If the field already displays a fully qualified class path, you can accept that variable definition by completing the rest of this wizard page and clicking **Next**.

h. Use the Native library path field to specify additional class files that your JDBC driver might require to function properly on your WebSphere Application Server platform. Type the full directory path name of these class files.

i. Click **Next** . You now see the page entitled Enter database specific properties for the data source.

10. Complete all of the fields on the Enter database specific properties page.
    - Click **Use this data source in container managed persistence (CMP)** if container managed persistence (CMP) enterprise beans must access this data source.
    - Any other property fields that are displayed on this wizard page are specific to your database type. Consult the article “Vendor-specific data sources minimum required settings” on page 687 for information on these property settings. The article addresses both current and deprecated JDBC providers that are pre-defined in WebSphere Application Server.

    User-defined data sources: This wizard page does not display additional property fields for data sources that correspond with your user-defined JDBC providers. However, from the JDBC driver class files that you installed, Application Server can generally extract the necessary data source property names. Application Server defines them as data source *custom* properties, displays them on a custom properties console page, and assigns them default values. Consult your database documentation about setting these properties and any other

requirements for your user-defined data source. After you create the data source, navigate to the corresponding custom properties collection page in the administrative console by clicking **Data sources** > *my_new_data_source* > **Custom properties**. Review the property default values and modify them if necessary.

11. Click **Finish** to save the configuration and exit the wizard. You now see the Data source collection page, which displays your new configuration in a table along with other data sources that are configured for the same scope.

You can override the default values for some required data source properties, as well as configure non-required properties. Consult the following topics:

- "Connection pool settings" on page 665
- "WebSphere Application Server data source properties" on page 728
- "J2EE resource provider or connection factory custom properties collection" on page 733

Data source collection:

Use this page to view configured data sources, which are the resources that provide connections to your relational database.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC** > **Data sources**
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources**

Version requirement: If you are using the Enterprise JavaBean (EJB) 1.0 specification and the Java Servlet 2.2 specification, you must use the **Data sources (WebSphere Application Server Version 4)** console page.

Name:

Specifies the display name of this data source.

Click a data source name to edit the data source configuration settings.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for this data source.

Data type String

Scope:

Specifies the scope of the JDBC provider that encapsulates the driver implementation classes to support this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that encapsulates the appropriate classes.

Description:

Specifies a text description of the data source.

Data type String

Category:

Specifies a string that you can use to classify or group a data source.

Data type String

Data source settings:

Use this page to edit the properties of a data source.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC** > **Data sources** > *data_source*
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources** > *data_source*

Version requirements: If your application uses an Enterprise JavaBean (EJB) 1.1 or a Java Servlet 2.2 module, you must use the **Data sources (WebSphere Application Server V4)** > *data_source* console page.

Test connection:

Activates the test connection service for validating application connections to the data source.

Before you click **Test connection**, set your data source properties and click **Apply**.

Scope:

Specifies the scope of the JDBC provider that supports this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that encapsulates the driver implementation classes to support this data source.

Name:

Specifies the display name for the data source.

Valid characters for this name include letters and numbers, but NOT most of the special characters. For example you can set this field to *Test Data Source*. But any name starting with a period (•) or containing special characters (\ / , : ; ″ * ? < > | = + & % ’ `) is not a valid name.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name.

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services associate names with locations, services, information, and resources.

Naming services provide name-to-object mappings. Directory services provide information on objects and the search tools required to locate those objects.

There are many naming and directory service implementations, and the interfaces to them vary. JNDI provides a common interface that is used to access the various naming and directory services.

For example, you can use the name *jdbc/markSection*.

If you leave this field blank a JNDI name is generated from the name of the data source. For example, a data source name of markSection generates a JNDI name of *jdbc/markSection*.

After you set this value, save it, and restart the server, you can see this string when you run the dump name space tool.

Data type String

Container-managed persistence:

Specifies if this data source is used for container-managed persistence of enterprise beans.

If this field is checked, a CMP Connector Factory that corresponds to this data source is created for the relational resource adapter.

Data type Checkbox
Default Enabled (The field is checked.)

Description:

Specifies a text description for the resource.

Data type String

Category:

Specifies a category string you can use to classify or group the resource.

Data type String

Data store helper class name:

Specifies the name of the DataStoreHelper implementation class that extends the capabilities of your selected JDBC driver implementation class to perform database-specific functions.

WebSphere Application Server provides a set of DataStoreHelper implementation classes for each of the JDBC provider drivers it supports. These implementation classes are in the package com.ibm.websphere.rsadapter. For example, if your JDBC provider is DB2, then your default DataStoreHelper class is com.ibm.websphere.rsadapter.DB2DataStoreHelper. The administrative console page you are viewing, however, might make multiple DataStoreHelper class names available to you in a drop-down list; be sure to select the one required by your database configuration. Otherwise, your application might not work correctly. If you want to use a DataStoreHelper other than those displayed in the drop-down list, select **Specify a user-defined DataStoreHelper** and type a fully qualified class name. Refer to the Information Center topic ″Example: Developing your own DataStoreHelper class.″

Data type Drop-down list or string (if **user-defined DataStoreHelper** is selected)

Component-managed Authentication Alias:

This alias is used for database authentication at run time.

The **Component-managed Authentication Alias** is only used when the application resource reference is using *res-auth = Application*.

If your database (for example, Cloudscape) does not support *user ID* and *password*, then do not set the alias in the component-managed authentication alias or container-managed authentication alias fields. Otherwise, you see the warning message in the system log to indicate that the user and password are not valid properties. (This message is only a warning message; the data source is still created successfully.)

If you do not set an alias (component-managed or otherwise), and your database requires the user ID and password to get a connection, then you receive an exception during run time.

Data type Drop-down list

Authentication Alias for XA Recovery:

This optional field is used to specify the authentication alias that should be used during XA recovery processing.

If the resource adapter does not support XA transactions, then this field will not be displayed. The default value will come from the selected alias for application authentication (if specified).

Use Component-managed Authentication Alias
Selecting this radio button specifies that the alias set for Component-managed Authentication is used at XA recovery time.

Data type Radio button

Specify:
Selecting this radio button enables you to choose an authentication alias from a drop-down list of configured aliases.

Data type Radio button

Container-managed Authentication Alias (deprecated):

Specifies authentication data (a string path converted to userid and password) for container-managed sign-on to the resource.

Note: Beginning with WebSphere Application Server Version 6.0, the container-managed authentication alias is superseded by the specification of a login configuration on the resource-reference mapping at deployment time, for components with *res-auth=Container*.

Select an alias from the list.

To define a new alias that is not displayed in the list:
- Click **Apply**. Under Related Items, you now see a listing for J2EE Connector Architecture (J2C) authentication data entries.

- Click **J2EE Connector Architecture (J2C) authentication data entries**.
- Click **New**.
- Define an alias.
- Click **OK**. The console now displays an alias collection page. This page contains a table that lists all of your configured aliases. Before the table, this page also displays the name of your connection factory.
- Click the name of your J2C connection factory. You now see the configuration page for the connection factory.
- Select the new alias in the Container-managed authentication alias list.
- Click **Apply**.

Data type Drop-down list

Mapping-Configuration Alias (deprecated):

Specifies the authentication alias for the Java Authentication and Authorization Service (JAAS) mapping configuration that is used by this connection factory.

Note: Beginning with WebSphere Application Server Version 6.0, the Mapping-Configuration Alias is superseded by the specification of a login configuration on the resource-reference mapping at deployment time, for components with *res-auth=Container*.

Click **Security** > **Secure administration, applications, and infrastructure** > **Java Authentication and Authorization Service** > **Application logins** and select an alias from the table.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations.

Data type Drop-down list

Important data source properties: These properties are specific to the data source that corresponds to your selected JDBC provider. They are either required by the data source, or are especially useful for the data source. You can find a complete list of the properties required for all supported JDBC providers in the topic ″Vendor-specific data sources minimum required settings″ in the Information Center.

WebSphere Application Server data source properties:

Use this page to set advanced WebSphere Application Server data source properties. These properties activate and configure services that WebSphere Application Server applies to data sources, to customize connection use within the application server. These properties do not affect the use of connections within the database.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC** > **Data sources** > *data_source* > **WebSphere Application Server connection properties**
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources** > *data_source* > **WebSphere Application Server connection properties**

Statement Cache Size:

Specifies the number of statements that can be cached per connection. WebSphere Application Server caches a statement after the user closes it.

The WebSphere Application Server data source optimizes the processing of *prepared statements* and *callable statements* by caching those statements that are not being used in an active connection. Both statement types help maximize the performance of transactions between your application and datastore.

- A prepared statement is a precompiled SQL statement that is stored in a `PreparedStatement` object. Application Server uses this object to run the SQL statement multiple times, as required by your application run time, with values that are determined by the run time.
- A callable statement is an SQL statement that contains a call to a stored procedure, which is a series of precompiled statements that perform a task and return a result. The statement is stored in the `CallableStatement` object. Application Server uses this object to run a stored procedure multiple times, as required by your application run time, with values that are determined by the run time.

If the statement cache is not large enough, useful entries are discarded to make room for new entries. To determine the largest value for your cache size to avoid any cache discards, add the number of uniquely prepared statements and callable statements (as determined by the *sql* string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible statements that can be cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. In general, the more statements your application has, the larger the cache should be.

Note: This statement cache size setting is different from WebSphere Application Server V4.0.x. In V4.0.x, the maximum number of possible prepared statements is cached for the data source within an application server. In V5 and higher, statement cache size is defined on a given physical connection.

You can also use the Tivoli Performance Viewer to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings. **Note:** The higher the statement cache, the more system resources are delayed. Therefore, if you set the number too high, you could lack resources because your system is not able to open that many prepared statements.

In test applications, tuning the statement cache improved throughput by 10-20%. However, because of potential resource limitations, this might not always be possible.

Data type	Integer
Default	Depends on the database. Most are 10. Informix versions 7.3, 9.2, 9.3, and 9.4, without the respective latest fixes, must be 0. A default of 0 means there is no cache statement.

Enable Multithreaded Access Detection: If checked, the application server detects the existence of access by multiple threads.

Enable Database Reauthentication: If checked, there cannot be an exact match on connections retrieved out of the WebSphere Application Server connection pool (that is, connection pool search criteria do not include user name and password). Instead, the reauthentication of connection is done in the *doConnectionSetupPerTransaction()* of the DataStoreHelper class. Note that WebSphere Application Server runtime does NOT provide connection reauthentication implementation. Therefore, when this box is checked you MUST extend the DataStoreHelper class to provide implementation of the *doConnectionSetupPerTransaction()* method where the reauthentication takes place. Failure to do that results in wrong connections being handed out to users. For more information, refer to the API documentation for *com.ibm.websphere.rsadapter.DataStoreHelper#doConnectionSetupPerTransaction(...)*.

Connection reauthentication can help improve performance by reducing the overhead of opening and closing connections, particularly for applications that always request connections with different user names and passwords.

Enable JMS One Phase Optimization Support: If checked, the application server allows JMS to get optimized connections from this data source. This property prevents JDBC applications from sharing connections with CMP applications.

Cached handles: Specifies whether the container tracks cached handles, which are connection handles that an application component holds active across transaction and method boundaries. Use this property for debugging purposes. Tracking handles can cause large performance overhead during runtime.

Log missing transaction context: Specifies whether the container issues an entry to the activity log when an application obtains a connection without a transaction context. These are exceptions to the J2EE programming model connection requirements.

PreTest existing pooled connections: If checked, the application server tries to connect to this data source before it attempts to send data to or receive data from this data source. If you select this property, you can specify how often, in seconds, the application server retries to make a connection if the initial attempt fails.

Retry interval: When **PreTest existing pooled connections** is checked, use this property to specify how long, in seconds, the application server waits before retrying to make a connection if the initial attempt fails.

Pretest new connections: Specifies whether the connection manager tests a new database connection that fulfills the initial connection request of an application. This operation can expose a database that returns bad connections. For example, bad connections can result from a database that fulfills connection requests before the database is completely functional.

Number of retries: Specifies the number of times you want to retest the initial connection to a database after the first pretest operation fails.

Retry interval: When **PreTest new connections** is checked, use this property to specify how long, in seconds, the application server waits before retrying to make a connection if the initial attempt fails.

PreTest SQL String:

Specifies the string of data that the application server sends to the database to test the connection. Use a query that is likely to have little impact on performance.

Data type Integer

Data sources (WebSphere Application Server V4):

Use this page to view the settings of a WebSphere Application Server Version 4.0 data source.

These data sources use the WebSphere Application Server Version 4.0 Connection Manager architecture. All EJB 1.1 modules must use a WebSphere Application Server Version 4.0 data source.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC**> **Data sources (WebSphere Application Server V4)**
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources (WebSphere Application Server V4)**

Name:

Specifies a text identifier of the data source.

Data type String

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source.

Data type String

Scope:

Specifies the scope of the JDBC provider that encapsulates the driver implementation classes to support this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that encapsulates the appropriate classes.

Description:

Specifies a text description of the data source.

Data type String

Category:

Specifies a text string that you can use to classify or group the data source.

Data type String

Data source (WebSphere Application Server Version 4) settings:

Use this page to create a Version 4.0 style data source. This data source uses the WebSphere Application Server Version 4.0 connection manager architecture. All of your EJB1.x modules must use this data source.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC**> **Data sources (WebSphere Application Server V4)** > *data_source*
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources (WebSphere Application Server V4)** > *data_source*

Scope:

Specifies the scope of the JDBC provider that encapsulates the driver implementation classes to support this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that WebSphere Application Server uses for this data source.

The list shows all of the existing JDBC providers that are defined at the relevant scope. Select one from the list if you want to use an existing JDBC provider as Provider.

Create New Provider:

Provides the option of configuring a new JDBC provider for the new data source.

Create New Provider is displayed only when you create, rather than edit, a data source.

Clicking **Create New Provider** causes the **Create a new JDBC provider** wizard to launch. Complete all of the wizard panels, then click **Finish**. The administrative console now displays the Data sources (WebSphere Application Server V4) configuration page again, where you see the new JDBC provider name in the Provider field.

Name:

Specifies the display name for the resource.

For example, you can set this field to *Test Data Source*.

Data type String

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name.

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services associate names with locations, services, information, and resources.

Naming services provide name-to-object mappings. Directory services provide information on objects and the search tools required to locate those objects.

There are many naming and directory service implementations, and the interfaces to them vary. JNDI provides a common interface that is used to access the various naming and directory services.

For example, you can use the name *jdbc/markSection*.

If you leave this field blank a JNDI name is generated from the name of the data source. For example, a data source name of markSection generates a JNDI name of *jdbc/markSection*.

After you set this value, save it, and restart the server, you can see this string when you run the dump name space tool.

Data type String

Description:

Specifies a text description for the resource.

Data type String

Category:

Specifies a category string that you can use to classify or group the resource.

Data type String

Database Name:

Specifies the name of the database that this data source accesses.

For example, you can call the database *SAMPLE*.

Data type String

Default User ID:

Specifies the user name to use for connecting to the database.

For example, you can use the ID *db2admin*.

Data type String

Default Password:

Specifies the password used for connecting to the database.

For example, you can use the password *db2admin*.

Data type String

J2EE resource provider or connection factory custom properties collection:

Use this page to view the custom properties of a J2EE resource provider or connection factory.

You can configure custom property collections for numerous resource types. According to the resource type with which a collection is associated, your ability to add, delete, and modify individual properties and settings varies. Begin the configuration process by clicking on the *Required* field to sort those column values in descending order. All of the required (true) values are then sorted at the beginning of the page. Be sure to set all required properties.

Deprecations: The following list displays three custom properties that are deprecated in WebSphere Application Server Version 6.10. The product now offers these properties as pre-configured options, which are the replacement properties in the following list. To avoid runtime error messages, permanently disable the original custom properties by deleting them from the table on this administrative console page.

- dbFailOverEnabled -- replaced by **Pretest new connection**
- validateNewConnection -- replaced by **Number of retries**
- connRetriesDuringDBFailover -- replaced by **Retry interval**

To set the replacement properties, click **Data sources** > *my_data_source* > **WebSphere Application Server data source properties**.

Name:

Specifies the property name.

You must ensure that the resource provider has the setting for this name.

Data type String

Value:

Specifies the property value.

Data type Variable; see “Custom property settings” for more information.

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Required:

Specifies whether this property is required for the resource provider.

Data type Boolean or Check box

Custom property settings:

Use this page to specify the attributes of custom properties that might be required for resource providers and resource factories.

According to the resource type with which a property collection is associated, your ability to modify individual property settings varies. Therefore, consider the following descriptions as a general reference for custom property settings. (The administrative console page that you are using to configure your custom property may only allow you to modify a subset of the following settings.)

Deprecations: The following list displays three custom properties that are deprecated in WebSphere Application Server Version 6.10. The product now offers these properties as pre-configured options, which are the replacement properties in the following list.

- dbFailOverEnabled -- replaced by **Pretest new connection**
- validateNewConnection -- replaced by **Number of retries**
- connRetriesDuringDBFailover -- replaced by **Retry interval**

Set the replacement properties on the WebSphere Application Server data source properties console page. Click **Data sources** > *my_data_source* > **WebSphere Application Server data source properties**.

Required:

Specifies properties that are required for this resource.

Data type Check box

Name:

Specifies the name associated with this property (PortNumber, ConnectionURL, etc).

Data type String

Value:

Specifies the value associated with this property in this property set.

Data type Determined by the **Type** setting, which you select from a drop-down list. If the type is `java.lang.String` then the value is of type String; if the type is `java.lang.Integer`, then the value is of type Integer; and so on.

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Type:

Specifies the fully qualified Java data type of this property .

There are specific types that are valid:
- java.lang.Boolean
- java.lang.String
- java.lang.Integer
- java.lang.Double
- java.lang.Byte
- java.lang.Short
- java.lang.Long
- java.lang.Float
- java.lang.Character

Data type Drop-down list

Custom Properties (Version 4) collection:

Use this page to view properties for a WebSphere Application Server Version 4.0 data source.

You can access this administrative console page in one of two ways:
- **Resources** > **JDBC** > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Custom Properties**
- **Resources** > **JDBC** > **JDBC Providers** > *JDBC_provider* > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Custom Properties**

Name:

Specifies the name of the custom property

Data type String

Value:

Specifies the value of the custom property.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Required:

Specifies properties that are required for this resource.

Data type String

Custom property (Version 4) settings:

Use this page to add properties for a WebSphere Application Server Version 4.0 data source.

To view this administrative console page, click **Resources** > **JDBC** > **JDBC Providers**> *JDBC_provider* > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Custom Properties** > *custom_property*.

Scope:

Specifies the level to which this resource definition is visible -- the cell, node, or server level.

Resources such as JDBC providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the cell level, all users in that cell can look up and use that data source, which is unique within that cell. However, resource property settings are local to each server in the cell. For example, if you define *max connections* to 10, then each server in that cell can have 10 connections.

Cell The most general scope. Resources defined at the cell scope are visible from all nodes and servers, unless they are overridden. To view resources defined in the cell scope, do not specify a server or a node name in the scope selection form.

Node The default scope for most resource types. Resources defined at the node scope override any duplicates defined at the cell scope and are visible to all servers on the same node, unless they are overridden at a server scope on that node. To view resources defined in a node scope, do not specify a server, but select a node name in the scope selection form.

Server
The most specific scope for defining resources. Resources defined at the server scope override any duplicate resource definitions defined at the cell scope or parent node scope and are visible only to a specific server. To view resources defined in a server scope, specify a server name as well as a node name in the scope selection form.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

Data type String

Required:

Specifies properties that are required for this resource.

Data type Check box

Name:

Specifies the name associated with this property (PortNumber, ConnectionURL, etc).

Data type String

Value:

Specifies the value associated with this property in this property set.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Type:

Specifies the fully qualified Java type of this property (java.lang.Integer, java.lang.Byte).

Data type String

Creating and configuring a JDBC provider and data source using the Java Management Extensions API

If your application requires access to a JDBC connection pool from a J2EE 1.3 or 1.4 level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively. Alternatively, you can use the JMX API in combination with the WSadmin - scripting tool.

Note: Use the JMX API to create only data sources for which the product does *not* provide a template. For every JDBC provider WebSphere Application Server supports, the product provides a corresponding data source template. You can create supported providers and associated data sources through the administrative console, or by using the WSadmin - scripting tool. For a complete list of supported JDBC providers (and therefore a complete list of data sources that must be created using a template), refer to the topic "Vendor-specific data sources minimum required settings" on page 687.

These steps outline the general procedure for using the JMX API to create a JDBC provider and data source, on WebSphere Application Server running on Windows platforms:

1. Put the appropriate JAR files in your classpath.

   You need two JAR files in your classpath -- `wsexception.jar` and `wasjmx.jar`.

   The following command is an example for setting your classpath:

   ```
   set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;D:\WebSphere\AppServer\lib\wasjmx.jar
   ```
2. Look up the host and get an administration client handle.
3. Get a configuration service handle.
4. Update the `resource.xml` file using the configuration service as desired.
   a. Add a JDBC provider.
   b. Add the data source.
   c. Add the connection factory. This step is necessary only for data sources that must support container-managed persistence.

5. Reload the resource.xml file to bind the newly created data source into the JNDI namespace. Perform this step if you want to use the newly created data source right away without restarting the application server.
   a. Locate the DataSourceConfigHelper MBean using the name.
   b. Put together the signature and parameters for the call.
   c. Invoke the reload() call.
6. **Attention:** If you modify the class path or native library path of an existing JDBC provider, you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

Example: Using the Java Management Extensions API to create a JDBC driver and data source for container-managed persistence:

```
//
// "This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines
// Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;


/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for CMP use.
 *
* We need following to run
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
      D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
  */
public class CreateDataSourceCMP {

   String dsName = "markSection"; // ds display name , also jndi name and
    CF name
   String dbName = "SECTION";     // database name
   String authDataAlias = "db2admin"; // an authentication data alias
   String uid = "db2admin";           // userid
   String pw  = "db2admin";           //  password
   String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the db driver




   /**
    * Main method.
    */
   public static void main(String[] args) {
      CreateDataSourceCMP cds = new CreateDataSourceCMP();

      try {
         cds.run(args);
      } catch (com.ibm.ws.exception.WsException ex) {
         System.out.println("Caught this " + ex   );
         ex.printStackTrace();
```

```
         //ex.getCause().printStackTrace();
      } catch (Exception ex) {
         System.out.println("Caught this " + ex );
         ex.printStackTrace();
      }
   }


   /**
    * This method creates the datasource using JMX.
* The datasource created here is only written into resources.xml.
    * It is not bound into namespace until the server is restarted, or
       an application started
    */
   public void run(String[] args) throws Exception {

      try {
         // Initialize the AdminClient.
         Properties adminProps = new Properties();
         adminProps.setProperty(AdminClient.CONNECTOR_TYPE,
          AdminClient.CONNECTOR_TYPE_SOAP);
         adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
         adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
         AdminClient adminClient =
              AdminClientFactory.createAdminClient(adminProps);

         // Get the ConfigService implementation.
         com.ibm.websphere.management.configservice.ConfigServiceProxy
           configService =
         new com.ibm.websphere.management.configservice.
                ConfigServiceProxy(adminClient);

         Session session = new Session();

         // Use this group to add to the node scoped resource.xml.
         ObjectName node1 = ConfigServiceHelper.createObjectName(null,
                "Node", null);
         ObjectName[] matches = configService.queryConfigObjects(session,
              null, node1, null);
         node1 = matches[0];     // use the first node found

         // Use this group to add to the server1 scoped resource.xml.
         ObjectName server1 = ConfigServiceHelper.createObjectName(null,
                 "Server", "server1");
         matches = configService.queryConfigObjects(session, null, server1,
                null);
         server1 = matches[0];   // use the first server found

         // Create the JDBCProvider
         String providerName = "DB2 JDBC Provider (XA)";
         System.out.println("Creating JDBCProvider " + providerName );

         // Prepare the attribute list
         AttributeList provAttrs = new AttributeList();
         provAttrs.add(new Attribute("name", providerName));
         provAttrs.add(new Attribute("implementationClassName",
                 "COM.ibm.db2.jdbc.DB2XADataSource"));
         provAttrs.add(new Attribute("description","DB2 JDBC2-compliant
                      XA Driver"));

         //create it
         ObjectName jdbcProv = configService.createConfigData(session,node1,
                 "JDBCProvider", "resources.jdbc:JDBCProvider",provAttrs);
         // now plug in the classpath
configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);
```

```
// Search for RRA so we can link it to the datasource
         ObjectName rra = ConfigServiceHelper.createObjectName(null,
                 "J2CResourceAdapter", null);
         matches = configService.queryConfigObjects(session, node1, rra,
                  null);
         rra = matches[0]; // use the first J2CResourceAdapter segment for
                  builtin_rra

         // Prepare the attribute list
         AttributeList dsAttrs = new AttributeList();
         dsAttrs.add(new Attribute("name", dsName));
         dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
         dsAttrs.add(new Attribute("datasourceHelperClassname",
              "com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
         dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
         dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
         // this is where we make the link to "builtin_rra"
         dsAttrs.add(new Attribute("description", "JDBC Datasource for
             mark section CMP 2.0 test"));
         dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

         // Create the datasource
         System.out.println("  **  Creating datasource");
         ObjectName dataSource =
             configService.createConfigData(session,jdbcProv,"DataSource",
               "resources.jdbc:DataSource",dsAttrs);

         // Add a propertySet.
         AttributeList propSetAttrs = new AttributeList();
         ObjectName resourcePropertySet =
           configService.createConfigData(session,dataSource,"propertySet",
               "",propSetAttrs);

         // Add resourceProperty databaseName
         AttributeList propAttrs1 = new AttributeList();
         propAttrs1.add(new Attribute("name", "databaseName"));
         propAttrs1.add(new Attribute("type", "java.lang.String"));
         propAttrs1.add(new Attribute("value", dbName));

         configService.addElement(session,resourcePropertySet,
              "resourceProperties",propAttrs1,-1);


// Now Create the corresponding J2CResourceAdapter Connection Factory object.
         ObjectName jra = ConfigServiceHelper.createObjectName(null,
             "J2CResourceAdapter",null);

         // Get all the J2CResourceAdapter, and I want to add my datasource
         System.out.println("  **  Get all J2CResourceAdapter's");
         ObjectName[] jras = configService.queryConfigObjects(session, node1,
                jra, null);

         int i=0;

         for (;i< jras.length;i++) {
            System.out.println(ConfigServiceHelper.getConfigDataType(jras[i])+
                    " " + i + " = "
                              + jras[i].getKeyProperty(SystemAttributes.
                                _WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
                              + "\nFrom scope ="
                              + jras[i].getKeyProperty(SystemAttributes.
                                _WEBSPHERE_CONFIG_DATA_ID));
            // quit on the first builtin_rra
            if (jras[i].getKeyProperty(SystemAttributes.
                              _WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
                .equals("WebSphere Relational Resource Adapter")) {
               break;
```

```
            }
         }


         if (i >= jras.length) {
            System.out.println("Did not find builtin_rra J2CResourceAdapter
                object creating CF anyways" );
         } else {
            System.out.println("Found builtin_rra J2CResourceAdapter
                   object at index " + i + " creating CF" );
         }

         // Prepare the attribute list
         AttributeList cfAttrs = new AttributeList();
         cfAttrs.add(new Attribute("name", dsName + "_CF"));
         cfAttrs.add(new Attribute("authMechanismPreference","BASIC_PASSWORD"));
         cfAttrs.add(new Attribute("authDataAlias",authDataAlias));
         cfAttrs.add(new Attribute("cmpDatasource", dataSource ));
         // this is where we make the link to DataSource's xmi:id
         ObjectName cf = configService.createConfigData(session,jras[i],
         "CMPConnectorFactory", "resources.jdbc:CMPConnectorFactory",cfAttrs);


         // ===== start Security section
         System.out.println("Creating an authorization data alias " +
            authDataAlias);

// Find the parent security object
         ObjectName security = ConfigServiceHelper.createObjectName(null,
              "Security", null);
         ObjectName[] securityName = configService.queryConfigObjects(session,
                null, security, null);
         security=securityName[0];

         // Prepare the attribute list
         AttributeList authDataAttrs = new AttributeList();
         authDataAttrs.add(new Attribute("alias", authDataAlias));
         authDataAttrs.add(new Attribute("userId", uid));
         authDataAttrs.add(new Attribute("password", pw));
         authDataAttrs.add(new Attribute("description","Auto created alias
             for datasource"));

         //create it
         ObjectName authDataEntry = configService.createConfigData
            (session,security,"authDataEntries", "JAASAuthData",authDataAttrs);
         // ===== end Security section

         // Save the session
         System.out.println("Saving session" );
         configService.save(session, false);

         // reload resources.xml to bind the new datasource into the name space
         reload(adminClient,true);
      } catch (Exception ex) {
         ex.printStackTrace(System.out);
         throw ex;
      }
   }

   /**
    * Get the DataSourceConfigHelperMbean and call reload() on it
    *
    * @param adminClient
    * @param verbose true - print messages to stdout
    */
   public void reload(AdminClient adminClient,boolean verbose) {
      if (verbose) {
```

```
         System.out.println("Finding the Mbean to call reload()");
      }

      // First get the  Mbean
      ObjectName handle = null;
      try {
         ObjectName queryName = new ObjectName("WebSphere:type=
                 DataSourceCfgHelper,*");
         Set s = adminClient.queryNames(queryName, null);
         Iterator iter = s.iterator();
         if (iter.hasNext()) handle = (ObjectName)iter.next();
      } catch (MalformedObjectNameException mone) {
         System.out.println("Check the program variable queryName" + mone);
      } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
         System.out.println("Cannot connect to the application server" + ce);
      }

      if (verbose) {
         System.out.println("Calling reload()");
      }
      Object result = null;
      try {
         result = adminClient.invoke(handle, "reload", new Object[] {},
                new String[] {});
      } catch (MBeanException mbe) {
         if (verbose) {
            System.out.println("\tMbean Exception calling reload" + mbe);
         }
      } catch (InstanceNotFoundException infe) {
         System.out.println("Cannot find reload ");
      } catch (Exception ex) {
         System.out.println("Exception occurred calling reload()"  + ex);
      }

      if (result==null && verbose) {
         System.out.println("OK reload()");
      }
   }

}
```

Example: Using the Java Management Extensions API to create a JDBC driver and data source for bean-managed persistence, session beans, or servlets:

```
//
// "This program may be used, executed, copied, modified and distributed without royalty for the
// purpose of developing, using, marketing, or distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;


/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for BMP use.
 *
 * We need following to run
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
```

```
 * D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
 */
public class CreateDataSourceBMP {

   String dsName = "markSection"; // ds display name , also jndi name and CF name
   String dbName = "SECTION";     // database name
   String authDataAlias = "db2admin"; // an authentication data alias
   String uid = "db2admin";           // userid
   String pw  = "db2admin";           //  password
   String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the db driver




   /**
    * Main method.
    */
   public static void main(String[] args) {
      CreateDataSourceBMP cds = new CreateDataSourceBMP();

      try {
         cds.run(args);
      } catch (com.ibm.ws.exception.WsException ex) {
         System.out.println("Caught this " + ex   );
         ex.printStackTrace();
         //ex.getCause().printStackTrace();
      } catch (Exception ex) {
         System.out.println("Caught this " + ex );
         ex.printStackTrace();
      }
   }


   /**
    * This method creates the datasource using JMX.
    *
    * The datasource created here is only written into resources.xml.
    * It is not bound into namespace until the server is restarted, or an application started
    */
   public void run(String[] args) throws Exception {

      try {
         // Initialize the AdminClient.
         Properties adminProps = new Properties();
         adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
         adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
         adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
         AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

         // Get the ConfigService implementation.
         com.ibm.websphere.management.configservice.ConfigServiceProxy configService =
         new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

         Session session = new Session();

         // Use this group to add to the node scoped resource.xml.
         ObjectName node1 = ConfigServiceHelper.createObjectName(null, "Node", null);
         ObjectName[] matches = configService.queryConfigObjects(session, null, node1, null);
         node1 = matches[0];     // use the first node found

         // Use this group to add to the server1 scoped resource.xml.
         ObjectName server1 = ConfigServiceHelper.createObjectName(null, "Server", "server1");
         matches = configService.queryConfigObjects(session, null, server1, null);
         server1 = matches[0];   // use the first server found

         // Create the JDBCProvider
         String providerName = "DB2 JDBC Provider (XA)";
```

```
     System.out.println("Creating JDBCProvider " + providerName );

     // Prepare the attribute list
     AttributeList provAttrs = new AttributeList();
     provAttrs.add(new Attribute("name", providerName));
     provAttrs.add(new Attribute("implementationClassName", "COM.ibm.db2.jdbc.DB2XADataSource"));
     provAttrs.add(new Attribute("description","DB2 JDBC2-compliant XA Driver"));

     //create it
     ObjectName jdbcProv = configService.createConfigData(session,node1,"JDBCProvider",
"resources.jdbc:JDBCProvider",provAttrs);
     // now plug in the classpath
     configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);




     // Search for RRA so we can link it to the datasource
     ObjectName rra = ConfigServiceHelper.createObjectName(null, "J2CResourceAdapter", null);
     matches = configService.queryConfigObjects(session, node1, rra, null);
     rra = matches[0]; // use the first J2CResourceAdapter segment for builtin_rra

     // Prepare the attribute list
     AttributeList dsAttrs = new AttributeList();
     dsAttrs.add(new Attribute("name", dsName));
     dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
     dsAttrs.add(new Attribute("datasourceHelperClassname","com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
     dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
     dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
// this is where we make the link to "builtin_rra"
     dsAttrs.add(new Attribute("description", "JDBC Datasource for mark section CMP 2.0 test"));
     dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

     // Create the datasource
     System.out.println("  **  Creating datasource");
     ObjectName dataSource = configService.createConfigData(session,jdbcProv,"DataSource",
"resources.jdbc:DataSource",dsAttrs);

     // Add a propertySet.
     AttributeList propSetAttrs = new AttributeList();
     ObjectName resourcePropertySet =configService.createConfigData(session,dataSource,
"propertySet","",propSetAttrs);

     // Add resourceProperty databaseName
     AttributeList propAttrs1 = new AttributeList();
     propAttrs1.add(new Attribute("name", "databaseName"));
     propAttrs1.add(new Attribute("type", "java.lang.String"));
     propAttrs1.add(new Attribute("value", dbName));

     configService.addElement(session,resourcePropertySet,"resourceProperties",propAttrs1,-1);


     // ===== start Security section
     System.out.println("Creating an authorization data alias " + authDataAlias);

     // Find the parent security object
     ObjectName security = ConfigServiceHelper.createObjectName(null, "Security", null);
     ObjectName[] securityName = configService.queryConfigObjects(session, null, security, null);
     security=securityName[0];

     // Prepare the attribute list
     AttributeList authDataAttrs = new AttributeList();
     authDataAttrs.add(new Attribute("alias", authDataAlias));
     authDataAttrs.add(new Attribute("userId", uid));
     authDataAttrs.add(new Attribute("password", pw));
     authDataAttrs.add(new Attribute("description","Auto created alias for datasource"));
```

```
            //create it
            ObjectName authDataEntry = configService.createConfigData(session,security,"authDataEntries",
     "JAASAuthData",authDataAttrs);
            // ===== end Security section

            // Save the session
            System.out.println("Saving session" );
            configService.save(session, false);

            // reload resources.xml
            reload(adminClient,true);

         } catch (Exception ex) {
            ex.printStackTrace(System.out);
            throw ex;
         }
      }

      /**
       * Get the DataSourceConfigHelperMbean and call reload() on it
       *
       * @param adminClient
       * @param verbose true - print messages to stdout
       */
      public void reload(AdminClient adminClient,boolean verbose) {
         if (verbose) {
            System.out.println("Finding the Mbean to call reload()");
         }

         // First get the  Mbean
         ObjectName handle = null;
         try {
            ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
            Set s = adminClient.queryNames(queryName, null);
            Iterator iter = s.iterator();
            if (iter.hasNext()) handle = (ObjectName)iter.next();
         } catch (MalformedObjectNameException mone) {
            System.out.println("Check the program variable queryName" + mone);
         } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot connect to the application server" + ce);
         }

         if (verbose) {
            System.out.println("Calling reload()");
         }
         Object result = null;
         try {
            result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
         } catch (MBeanException mbe) {
            if (verbose) {
               System.out.println("\tMbean Exception calling reload" + mbe);
            }
         } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find reload ");
         } catch (Exception ex) {
            System.out.println("Exception occurred calling reload()"  + ex);
         }

         if (result==null && verbose) {
            System.out.println("OK reload()");
         }
      }
   }
```

Example: Creating a JDBC provider and data source using Java Management Extensions API and the scripting tool: The following code is a JACL (WSadmin - scripting tool) script used to create a data source. Use this script to create only data sources for which the product does *not* provide a template. For every JDBC provider WebSphere Application Server supports, the product provides a corresponding data source template. See the topic Creating configuration objects using the wsadmin tool for instructions on how to use the `createUsingTemplate` command to establish these data sources. For a complete list of supported JDBC providers (and therefore a complete list of data sources that must be created using a template), refer to the topic "Vendor-specific data sources minimum required settings" on page 687.

This script sets up the following sample JDBC objects:
- Creates a data source *fvtDS_1*
- Creates a 4.0 data source *fvtDS_3*
- Creates a container-managed persistence (CMP) data source linked to *fvtDS_1*

Attention: If you later modify the class path or native library path of the JDBC provider associated with your data source, you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

```
#AWE --  Set up XA DB2 data sources, both Version 4.0 and J2EE Connector architecture (JCA)-compliant data sources


#UPDATE THESE VALUES:
#The classpath that will be used by your database driver
set driverClassPath "c:/sqllib/java/db2java.zip"

set server "server1"

set fvtbase "c:/wssb/fvtbase"

#Users and passwords..
set defaultUser1 "dbuser1"
set defaultPassword1 "dbpwd1"
set aliasName "alias1"

set databaseName1 "jtest1"
set databaseName2 "jtest2"
#END OF UPDATES

puts "Add an alias alias1"
set cell [$AdminControl getCell]
set sec [$AdminConfig getid /Cell:$cell/Security:/]

#-------------------------------------------------------
# Create a JAASAuthData object for component-managed authentication
#-------------------------------------------------------
puts "create JAASAuthData object for alias1"

set alias_attr    [list alias $aliasName]
set desc_attr     [list description "Alias 1"]
set userid_attr   [list userId $defaultUser1]
set password_attr [list password $defaultPassword1]
set attrs [list $alias_attr $desc_attr $userid_attr $password_attr]

set authdata [$AdminConfig  create JAASAuthData $sec $attrs]
$AdminConfig save

puts "Installing DB2 datasource for XA"

puts "Finding the old JDBCProvider.."
#Remove the old jdbc provider...
set jps [$AdminConfig list JDBCProvider]
foreach jp $jps {
 set jpname [lindex [lindex [$AdminConfig show $jp {name}] 0] 1]
 if {($jpname == "FVTProvider")} {
```

```
  puts "Removing old JDBC Provider"
  $AdminConfig remove $jp
  $AdminConfig save
 }
}


#Get the server name...
puts "Finding the server $server"
set servlist [$AdminConfig list Server]
set servsize [llength $servlist]
foreach srvr $servlist {
 set sname [lindex [lindex [$AdminConfig show $srvr {name}] 0] 1]
 if {($sname == $server)} {
                puts "Found server $srvr"
  set serv $srvr
 }
}

set desiredNodeName "myNode"
puts "Finding the Node"
set nodelist [$AdminConfig list Node]
foreach node $nodelist {
  set nodename [lindex [lindex [$AdminConfig show $node {name}] 0] 1]
  if {($nodename == $desiredNodeName)} {
    puts "node = $node"
    break
  }
}

puts "Finding the Resource Adapter"
set ralist [$AdminConfig list J2CResourceAdapter $node]
set ralistlen [llength $ralist]
foreach ra $ralist {
  set raname [lindex [lindex [$AdminConfig show $ra {name}] 0] 1]
  if {($raname == "WebSphere Relational Resource Adapter")} {
    set rsadapter $ra
  }
}

#Now create a JDBC Provider for the data sources
puts "Creating the provider for COM.ibm.db2.jdbc.DB2XADataSource"
set attrs1 [subst {{classpath $driverClassPath}
 {implementationClassName COM.ibm.db2.jdbc.DB2XADataSource}{name "FVTProvider2"}
 {description "DB2 JDBC Provider"} {xa "true"}}]
set provider1 [$AdminConfig create JDBCProvider $serv $attrs1]

#Create the first data source
puts "Creating the datasource fvtDS_1"
set attrs2 [subst {{name fvtDS_1} {description "FVT DataSource 1"}}]
set ds1 [$AdminConfig create DataSource $provider1  $attrs2]

#Set the properties for the data source.
set propSet1 [$AdminConfig create J2EEResourcePropertySet $ds1 {}]

set attrs3 [subst {{name databaseName} {type java.lang.String} {value $databaseName1}}]
$AdminConfig create J2EEResourceProperty $propSet1 $attrs3

set attrs10 [subst {{jndiName jdbc/fvtDS_1} {statementCacheSize 10}
 {datasourceHelperClassname com.ibm.websphere.rsadapter.DB2DataStoreHelper}
 {relationalResourceAdapter {$rsadapter}} {authMechanismPreference "BASIC_PASSWORD"}
 {authDataAlias  $aliasName}}]
$AdminConfig modify $ds1 $attrs10


#Create the connection pool object...
$AdminConfig create ConnectionPool $ds1 {{connectionTimeout 1000}
```

```
 {maxConnections 30} {minConnections 1} {agedTimeout 1000}
 {reapTime 2000} {unusedTimeout 3000} }


#Now create the 4.0 data sources..
puts "Creating the 4.0 datasource fvtDS_3"
set ds3 [$AdminConfig create WAS40DataSource $provider1 {{name fvtDS_3} {description "FVT 4.0 DataSource"}}]

#Set the properties on the data source
set propSet3 [$AdminConfig create J2EEResourcePropertySet $ds3 {}]

#These attributes should be the same as fvtDS_1
set attrs4 [subst {{name user} {type java.lang.String} {value $defaultUser1}}]
set attrs5 [subst {{name password} {type java.lang.String} {value $defaultPassword1}}]
$AdminConfig create J2EEResourceProperty $propSet3 $attrs3
$AdminConfig create J2EEResourceProperty $propSet3 $attrs4
$AdminConfig create J2EEResourceProperty $propSet3 $attrs5
set attrs10 [subst {{jndiName jdbc/fvtDS_3} {databaseName $databaseName1}}]
$AdminConfig modify $ds3 $attrs10

$AdminConfig create WAS40ConnectionPool $ds3 {{orphanTimeout 3000} {connectionTimeout 1000}
 {minimumPoolSize 1} {maximumPoolSize 10} {idleTimeout 2000}}

#Now add a CMP connection factory for the JCA-compliant data source. This step is not necessary for
#Version 4 data sources, as they contain built-in CMP connection factories.
puts "Creating the CMP Connector Factory for fvtDS_1"
set attrs12 [subst {{name "FVT DS 1_CF"} {authMechanismPreference BASIC_PASSWORD}
 {cmpDatasource $ds1} {authDataAlias $aliasName}}]
set cf1 [$AdminConfig create CMPConnectorFactory $rsadapter $attrs12]

#Set the properties for the data source.
$AdminConfig create MappingModule $cf1 {{mappingConfigAlias "DefaultPrincipalMapping"} {authDataAlias "alias1"}}

$AdminConfig save
```

Verifying a connection

Many connection problems can be easily fixed by verifying some configuration parameters. This article provides a checklist of steps that you must complete to enable a successful connection. Click on the link for more information on a specific step.

If your connection is still not successful after completing these steps and reviewing the applicable information, check the SystemOut.log for warning or exception messages. Then use the technical support search function to find known problems.

1. Create the authentication data alias.
2. Create the JDBC provider.
3. Create a data source.
4. Save the data source.
5. If you created a new authentication alias, restart the server for which you need to verify connectivity.
6. Test the connection

   You can test your connection from the data source collection view or the data source details view. Access either view in the administrative console, and then select a connection from the list. Click the **Test Connection** button on the connection.

Test connection service

WebSphere Application Server provides a test connection service for testing connections to a data source.

If you associate your data sources with WebSphere variables, see the Configuring WebSphere variables topic to verify that you configure them correctly. A `variable cannot be found` exception results from attempted use of a data source that is invoked through an incorrectly defined variable.

Activating the test connection service

There are three ways to activate the test connection service: through the administrative console, the wsadmin tool, or a Java stand-alone program. Each process invokes the same methods on the same MBean.

Administrative console

WebSphere Application Server allows you to test a connection from the administrative console by simply pushing a button: the *Data source collection*, *Data source settings*, *Version 4 data source collection*, and *Version 4 data source settings* pages all have **Test Connection** buttons. After you define and save a data source, you can click this button to ensure that the parameters in the data source definition are correct. On the collection page, you can select several data sources and test them all at once. Note that there are certain conditions that must be met first. For more information, see Testing a connection with the administrative console.

WsAdmin tool

The wsadmin tool provides a scripting interface to a full range of WebSphere Application Server administration activities. Because the Test Connection functionality is implemented as a method on an MBean, and wsadmin can invoke MBean methods, wsadmin can be utilized to test connections to data sources. You have two options for testing a data source connection through wsadmin:

The *AdminControl* object of wsadmin has a testConnection operation that tests the configuration properties of a data source object. For information, see Testing a connection using wsadmin.

You can also test a connection by invoking the MBean operation. Use Example: Testing data source connection using wsadmin as a guide for this technique.

Java stand-alone program

Finally, you can test a connection by executing the `testConnection()` method on the `DataSourceCfgHelper` MBean. This method allows you to pass the configuration ID of the configured data source. The Java program connects to a running Java Management Extensions (JMX) server to access the MBean. In a base installation of Application Server, you connect to the JMX server running in the application server, usually on port 8880.

The return value from this invocation is either 0, a positive number, or an exception. 0 indicates that the operation completed successfully, with no warnings. A positive number indicates that the operation completed successfully, with the number of warnings. An exception indicates that the test of the connection failed.

You can find an example of this code in Example: Test a connection using testConnection(ConfigID).

Testing a connection with the administrative console

After you have defined and saved a data source, you can click the **Test Connection** button to ensure that the parameters in the data source definition are correct.

You can select multiple data sources on the data source collection page and test them as a group. Be sure that the following conditions are met before using the Test Connection button:

- If you are testing a connection using a WebSphere Application Server Version 4.0 type of data source, ensure that the *user* and *password* information is set.
- Designate variables appropriately.

If you used a WebSphere environment entry for the class path or other fields, such as *${DB2_JDBC_DRIVER_PATH}/db2java.zip*, make sure that you assign it a value in the *WebSphere Variables* page.

- Restart the application server after you define or edit WebSphere variables.
- Restart the application server after you create or edit an authentication alias for the data source.
- You can now test a connection to the data source. On the data source collection page in the administrative console, select the data source and click **Test Connection** .

  A Test Connection operation can have three different outcomes, each resulting in a different message being displayed in the messages panel of the page on which you press the Test Connection button.
  1. The test can complete successfully, meaning that a connection is successfully obtained to the database using the configured data source parameters. The resulting message states: Test Connection for data source *DataSourceName* on process *ProcessName* at node *NodeName* was successful.
  2. The test can complete successfully with warnings. This means that while a connection is successfully obtained to the database, warnings were issued. The resulting message states: Test Connection for data source *DataSourceName* on process *ProcessName* at node *NodeName* was successful with warning(s). View the JVM Logs for more details.

     The **View the JVM Logs** text is a hyperlink that takes you to the JVM Logs console screen for the process.
  3. The test can fail. A connection to the database with the configured parameters is not obtained. The resulting message states: Test Connection failed for data source *DataSourceName* on process *ProcessName* at node *NodeName* with the following exception: *ExceptionText*. View the JVM Logs for more details.

     Again, the text for **View the JVM Logs** is a hyperlink to the appropriate logs screen.

Testing a connection using wsadmin

The *AdminControl* object of wsadmin has a testConnection operation that tests the configuration properties of a data source object.

The testConnection operation takes a data source *configuration ID* as an argument.

Note: This invocation cannot accept user IDs and passwords that must be defined in the database itself. This invocation can only be used for databases that do not require a user ID and password to make a connection (such as DB2 on a Windows machine), or for data sources that have a component-managed or container-managed authentication alias set on the data source object.

1. Invoke the getid() method for your data source.
2. Set the value of the *configuration id* to a variable.

   ```
   set myds [$AdminConfig getid /JDBCProvider:mydriver/DataSource:mydatasrc/]
   ```

   where */JDBCProvider:mydriver/DataSource:mydatasrc/* is the data source you want to test. After you have the configuration ID of the data source, you can test the connection to the database.
3. Test the connection to the database.

   ```
   $AdminControl testConnection $myds
   ```

Example: Test a connection using testConnection(ConfigID): This program uses JMX to connect to a running server and invoke the *testConnection* method on the *DataSourceCfgHelper* MBean.

```
/**
 * Description
 * Resource adapter test program to make sure that the MBean interfaces work.
 * Following interfaces are tested
 *
 *  ---  testConnection()
 *
 *
 * We need following to run
```

```
 * C:\src>java -Djava.ext.dirs=C:\WebSphere\AppServer\lib;C:\WebSphere\AppServer\java\jre\lib\ext testDSGUI
 * must include jre for log.jar and mail.jar, else get class not found exception
 *
 *
 */

import java.util.Iterator;
import java.util.Locale;
import java.util.Properties;
import java.util.Set;

import javax.management.InstanceNotFoundException;
import javax.management.MBeanException;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.RuntimeMBeanException;
import javax.management.RuntimeOperationsException;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.ws.rsadapter.exceptions.DataStoreAdapterException;

public class testDSGUI {

//Use port 8880 for base installation or port 8879 for ND installation
String port = "8880";
// String port = "8879";
String host = "localhost";
final static boolean verbose = true;

// eg a configuration ID for DataSource declared at the node level for base
private static final String resURI = "cells/cat/nodes/cat:resources.xml#DataSource_1";

// eg a 4.0 DataSource declared at the node level for base
//    private static final String resURI = "cells/cat/nodes/cat:resources.xml#WAS40DataSource_1";

// eg Cloudscape DataSource declared at the server level for base
//private static final String resURI = "cells/cat/nodes/cat/servers/server1/resources.xml#DataSource_6";

// eg node level DataSource for ND
//private static final String resURI = "cells/catNetwork/nodes/cat:resources.xml#DataSource_1";

// eg server level DataSource for ND
//private static final String resURI = "cells/catNetwork/nodes/cat/servers/server1:resources.xml#DataSource_4";

// eg cell level DataSource for ND
//private static final String resURI = "cells/catNetwork:resources.xml#DataSource_1";

 public static void main(String[] args) {
  testDSGUI cds = new testDSGUI();
  cds.run(args);
 }

/**
 * This method tests the ResourceMbean.
 *
 * @param args
 * @exception Exception
 */
 public void run(String[] args) {

  try {

 System.out.println("Connecting to the application server.......");

        /*********************************************************************/
        /**    Initialize the AdminClient                                   */
```

```
       /*********************************************************************/
Properties adminProps = new Properties();
adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
adminProps.setProperty(AdminClient.CONNECTOR_HOST, host);
adminProps.setProperty(AdminClient.CONNECTOR_PORT, port);
AdminClient adminClient = null;
try {
    adminClient = AdminClientFactory.createAdminClient(adminProps);
} catch (com.ibm.websphere.management.exception.ConnectorException ce) {
      System.out.println("NLS: Cannot make a connection to the application server\n");
    ce.printStackTrace();
    System.exit(1);
}

      /*********************************************************************/
      /**    Locate the Mbean                                             */
      /*********************************************************************/
ObjectName handle = null;
try {
               // Send in a locator string
               // eg for a Baseinstallation this is enough
               ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");

               // for ND you need to specify which node/process you would like to test from
               // eg run in the server
//ObjectName queryName = new OjectName
    ("WebSphere:cell=catNetwork,node=cat,process=server1,type=DataSourceCfgHelper,*");
               // eg run in the node agent
//ObjectName queryName = new ObjectName
    ("WebSphere:cell=catNetwork,node=cat,process=nodeagent,type=DataSourceCfgHelper,*");
// eg run in the Deployment Manager
//ObjectName queryName = new ObjectName
   ("WebSphere:cell=catNetwork,node=catManager,process=dmgr,type=DataSourceCfgHelper,*");
 Set s = adminClient.queryNames(queryName, null);
 Iterator iter = s.iterator();
 while (iter.hasNext()) {
               // use the first MBean that is found
  handle = (ObjectName) iter.next();
 System.out.println("Found this ->" + handle);
 }
 if (handle == null) {
  System.out.println("NLS: Did not find this MBean>>" + queryName);
  System.exit(1);
 }
} catch (MalformedObjectNameException mone) {
 System.out.println("Check the program variable queryName" + mone);
} catch (com.ibm.websphere.management.exception.ConnectorException ce) {
 System.out.println("Cannot connect to the application server" + ce);
}

       /*********************************************************************/
       /**          Build parameters to pass to Mbean                      */
       /*********************************************************************/
String[] signature = { "java.lang.String" };
Object[] params = { resURI };
Object result = null;

      if (verbose) {
 System.out.println("\nTesting connection to the database using " + handle);
}

try {
              /*********************************************************************/
              /**  Start to test the connection to the database                */
              /*********************************************************************/
 result = adminClient.invoke(handle, "testConnection", params, signature);
} catch (MBeanException mbe) {
```

```
 // ****** all user exceptions come in here
 if (verbose) {
  Exception ex = mbe.getTargetException(); // this is the real exception from the Mbean
  System.out.println("\nNLS:Mbean Exception was received contains " + ex);
  ex.printStackTrace();
  System.exit(1);
 }
} catch (InstanceNotFoundException infe) {
 System.out.println("Cannot find " + infe);
} catch (RuntimeMBeanException rme) {
 Exception ex = rme.getTargetException();
 ex.printStackTrace(System.out);
 throw ex;
} catch (Exception ex) {
 System.out.println("\nUnexpected Exception occurred: " + ex);
 ex.printStackTrace();
}

        /*************************************************************************/
        /**  Process the result.  The result will be the number of warnings      */
        /**  issued.  A result of 0 indicates a successful connection with       */
        /**  no warnings.                                                        */
        /*************************************************************************/

//A result of 0 indicates a successful connection with no warnings.
System.out.println("Result= " + result);

 } catch (RuntimeOperationsException roe) {
Exception ex = roe.getTargetException();
ex.printStackTrace(System.out);
 } catch (Exception ex) {
System.out.println("General exception occurred");
ex.printStackTrace(System.out);
 }
}
}
```

Configuring data access for the Application Client

Configuring data access for the Application Client involves specifying the resource reference and associated database information required for data access. This specification is done as part of the assembly and deployment steps for the Application Client.

There are two tools needed to configure data sources used by J2EE application clients:

- An assembly tool such as the Application Server Toolkit (AST) or Rational Application Developer for defining the resource reference in the deployment descriptor; and
- The Application Client Resource Configuration Tool (ACRCT) for defining the connection to the database in the client deployment environment.

Data access from an application client uses the JDBC driver connection functions directly from the client side. It does not take advantage of the additional pooling support available in the WebSphere Application Server run time. Configuring data access for an application client does not require configuration of a JDBC provider and data source on the WebSphere Application Server server machine.

If you want to take advantage of the pooling and additional database functions provided by WebSphere Application Server, it is recommended that your client application utilize an enterprise bean running on the server side to perform data access.

Defining an application client resource reference using an assembly tool

1. Assemble your application client module as described in Assembling application clients.
2. Create a new resource reference:

a. In a Project Explorer view, right-click your application client module and click **Open With > Deployment Descriptor Editor**.
b. On the **References** tab, click **Add > Resource reference > Next**.
c. On the Resource Reference page, enter the **Name** of this resource reference. The Application Client for WebSphere Application Server run time uses this name for two purposes: to bind the object into the *java:comp/env* portion of the JNDI namespace, and to find client specific configuration information. If the code for the Application Client performs a lookup for *java:comp/env/jdbc/myDB*, the name of the resource reference should be *jdbc/myDB*.
d. For **Type**, select *javax.sql.DataSource* for JDBC connections.
e. For **Authentication**, select *Application* if your client application intends to provide authentication information. If the Application Client for WebSphere Application Server run time provides the authentication information (as configured by the Application Client Resource Configuration tool), select *Container*.
f. Ignore the **Sharing scope** setting; it is unused in an application client resource reference. All Application Client resources are not shared.
g. Click **Finish**.
h. Close the deployment descriptor and save your changes.

The JNDI name field appears under **WebSphere Bindings** after your add the reference.

Client configuration with the ACRCT

There are two client resources for you to configure in the Application Client Resource Configuration Tool (ACRCT) to enable data access from an application client: a data source provider and a data source.

Notes

Note: The following WebSphere objects, which can be bound into the server name space, are not supported on the client:
- Java 2 Connector (J2C) objects
- Connection manager objects

The WebSphere Application Server Client does not provide client database drivers. If your client application uses a database directly, rather than using an enterprise bean, you must provide the database drivers on the client machine. This action can involve contacting your database vendor to acquire client database driver code and licenses.

Instead of accessing the database directly, it is recommended that your client application use an enterprise bean. Accessing a database through an enterprise bean eliminates the need to have database drivers on the client machine because the database access is handled by the enterprise bean running on the WebSphere Application Server. Enterprise beans can also take advantage of the additional database functions provided by the WebSphere Application Server run time.

1. Configure a new data source provider as described in Configuring new data source providers. This provider describes the JDBC database implementation for your client application.
2. Enter the following information on the **General** tab:
   a. A **name** for this data source provider.
   b. **Optional:** A **description**.
   c. The **classpath** to the data source provider implementation classes or JAR files. This is optional if the implementation classes or JAR files are already in the class path configuration of the client.
   d. The name of the **implementation class**. For example, for DB2 this value is *COM.ibm.db2.jdbc.DB2DataSource*. Remember this class must implement the *javax.sql.DataSource* class. The ACRCT does not verify this class and you receive an error when you run your client application if the class does not implement *javax.sql.DataSource*.

Use the **Custom** tab to configure non-standard properties of the data source provider. This panel enables you to enter property-value pairs. During run time the *implementation class name* is created and any custom properties added on this panel are set on the newly created data source object using reflection. Any properties configured on this panel must have an appropriate set method on the data source class. For example, assume there is a property called *use2Phase* and its value should be 1. On the custom panel you enter the value *use2Phase* into the **name** column and the value *1* into the **value** column. The Application Client for WebSphere Application Server run time then uses reflection to find a property on the data source class called, typically *setUse2Phase* and call that method passing the value of 1. See your database product documentation for valid properties on your data source implementation.

3. Click **OK**.
4. Configure a new data source as described in Configuring new data sources for application clients. This describes the client properties of the database your client application uses.
5. Enter the following information on the **General** tab:
   a. A **Name**. This field is required and identifies a name for the Application Client Resource Configuration Tool to use. This name is **not** used by your client application program.
   b. **Optional:** A **description**.
   c. The **JNDI name**. This field is required and must match the value entered in the **Name** field on the Add Resource Reference page of the assembly tool. In the example above, set this value to *jdbc/myDB*.
   d. **Optional:** The **Database Name**.
   e. **Optional:** Your *userid* in the **User** field.
   f. **Optional:** Your *password* in the **Password** field. This password does not display.
   g. Your password again to confirm in the **Re-Enter password** field. Note: The **User** and **Password** fields are used only when the **Authentication** field on the Add Resource Reference page of the assembly tool is set to *Container*.

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** > **Enterprise Applications** > *application_name* > **Resource references**.

Guidelines for using this administrative console page:

- If your application uses any of the following resource types, you can set or reset their mapping configurations:
  - Mail session
  - URL configuration
  - Data source
  - 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
  - Generic JMS connection factory
- The page is comprised 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. 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 from the new page that is displayed, the Available Resources page. The Available Resources page shows all resources that are available mapping targets for your application references.
  3. Click **Apply**. The console displays the Resource 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.
- **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:
  - If you select **None**:
    1. Determine which resource 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**.
  - If you select **Default**:
    1. Determine which resources to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. You must apply this option to each resource individually if you want to designate different authentication data aliases. See the ″J2EE Connector security″ Information center topic for more information on the default mapping configuration.
    2. Select the appropriate table rows.
    3. Select **Use default method** 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**.
  - If you select **Custom login configuration**:
    1. Determine which resources to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the ″J2EE Connector security″ Information center topic 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.

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 contained 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 reference that is used in the enterprise bean, if applicable, and is declared in the deployment descriptor of the application module.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the resource that is the mapping target of the resource reference.

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.

Performing platform-specific tasks for JDBC access
This article provides miscellaneous tips for using supported databases. See also the related links.

Most of the following tasks are performed outside of the WebSphere Application Server administrative tools, using the database product tools. Always refer to the documentation that accompanies your database JDBC driver as the authoritative and complete source of driver information. The Supported hardware and software web page lists drivers that are supported by Application Server for specific platforms.

- **Enable Java Transaction API (JTA) drivers for DB2 on Windows NT systems** by following these steps:
  1. Bind the necessary packages to the database. From the **DB2 Command Line Processor** window, issue the following commands:

     ```
     db2=> connect to mydb2jta
     db2=> bind db2home\bnd\@db2cli.lst
     db2=> bind db2home\bnd\@db2ubind.lst
     db2=> disconnect mydb2jta
     ```

     where *mydb2jta* is the name of the database to enable for the JTA, and *db2home* is the DB2 root installation directory path (for example, *D:\ProgramFiles\SQLLIB\bnd\@db2cli.lst*).
  2. Specify the following settings when you use an IBM WebSphere Application Server administrative client (such as the administrative console) to configure a JDBC driver:
     - **Server class path** = `%DB2_ROOT%/Sqllib/java/db2java.zip`
     - **Implementation class name** = `COM.ibm.db2.jdbc.DB2XADataSource`
- **Enable JTA drivers on UNIX systems** by following these steps:
  1. Stop all DB2 services.
  2. Stop the IBM WebSphere Application Server administrative service.
  3. Stop any other processes that use the *db2java.zip* file.
  4. Make sure that you already enabled JDBC 2.0.

5. Start the DB2 services.
6. Bind the necessary packages to the database. From the DB2 command-line process or window, issue the following commands:

```
db2=> connect to mydb2jta
db2=> bind db2home\bnd\@db2cli.lst
db2=> bind db2home\bnd\@db2ubind.lst
db2=> disconnect mydb2jta
```

7. Specify the following settings when you use an IBM WebSphere Application Server administrative client (such as the administrative console) to configure a JDBC driver:
   - **Server class path** = `$INSTHOME/sqllib/java12/db2java.zip`

     For example, if *$INSTHOME* is `/home/test`, the path will be `/home/test/sqllib/java12/db2java.zip`
   - **Implementation class name** = `COM.ibm.db2.jdbc.DB2XADataSource`

- **Enable JTA drivers for Sybase products on AIX systems** by following these steps:
  1. Enable the Data Transaction Manager (DTM) by issuing these commands (one per line) at a command prompt:

```
isql -Usa -Ppassword -Sservername
sp_configure "enable DTM", 1
go
```

  2. Stop the Sybase Adaptive Server database and start it again.
  3. Grant the appropriate role authorization to the enterprise bean user at a command prompt:

```
isql -Usa -Ppassword -Sservername
grant role dtm_tm_role to EJB
go
```

- **Sybase Java Transaction API drivers on all platforms:** Do not use a Sybase Java Transaction API (JTA) connection in an enterprise bean method with an unspecified transaction context. A Sybase JTA connection does not support the local transaction mode. The implication is that you must use the Sybase JTA connection in a global transaction context.
- **Enable multiple session clients for local DB2 databases on AIX** by working around the limit for shared memory segments per process. AIX, by default, does not permit 32-bit applications to attach to more than 11 shared memory segments per process. Of these 11 shared segments, a maximum of 10 can be used for local DB2 connections. To use EXTSHM with DB2 and avoid stale connections when there are large numbers of session clients, do the following:
  - In DB2 client environment (that is, the WebSphere Application Server run-time environment in this case):

```
export EXTSHM=ON
```

  - In DB2 UDB Server environment:

```
export EXTSHM=ON
db2set DB2ENVLIST=EXTSHM
```

- **Source the `db2profile` script on UNIX systems** before starting WebSphere Application Server to host data access applications. Run the command

```
. ~db2inst1/sqllib/db2profile
```

  where *db2inst1* is the user created during DB2 installation.

Pretesting pooled connections to ensure validity

When a database fails, pooled connections that are not valid might exist in the free pool. This scenario is likely to occur when you have a failingConnectionOnly purge policy, which mandates that only failing connections be removed from the pool. Whether the remaining connections in the pool are valid varies with the cause of the failure. Connection pretesting is a way to test connections from the free pool before giving them to the client.

If your application uses pooled connections, you can enable the PreTest Connections feature in the administrative console to help prevent your application from obtaining connections that are no longer valid.

The feature is particularly useful for routine database outages. Because these outages are usually scheduled for periods of low use, connections to the database are likely to be in the free pool rather than in active use. Active connections are not pretested; pretesting impedes performance during normal operation. Pretesting ensures that users do not waste time trying to resume connections that became bad before the outage.

1. In the administrative console, click **Resources** > **JDBC providers**.
2. Select a provider and click **Data Sources** under Additional properties.
3. Select a data source and click **WebSphere Application Server data source properties** under Additional properties.
4. Select the **PreTest Connections** check box.
5. Type a value for the PreTest Connection Retry Interval, which is measured in seconds. This property determines the frequency with which a new connection request is made after a pretest operation fails.
6. Type a valid SQL statement for the PreTest SQL String. Use a reliable SQL command, with minimal performance impact; this statement is processed each time a connection is obtained from the free pool.

   For example, you might specify SELECT COUNT(*) from TESTTABLE. (For an Oracle database, use SELECT USER FROM DUAL.)

Passing client information to a database

Using a WebSphere Application Server API or trace function, you can pass unique client information on every connection that originates from the same data source.

Some databases, such as DB2, support a data source custom property that triggers your database servers to extract client information from WebSphere Application Server connections. (Consult the database documentation to see whether your product supports this capability and which property the product requires.) Be aware, however, that these properties introduce a very limited functionality in Application Server. Consequently, an application server connection manager incurs the following risky behaviors, which can result in the transfer of wrong client information to the database.

- The connection manager cannot change client information on the data source, or the connections obtained from that data source, dynamically.
- The connection manager must set the same client information on all connections that are obtained from that data source. For example, if you set `ApplicationName` as part of the data source `clientInformation` property, all connections from that data source have the same application name.

Application Server offers two methods of passing client information that provide the necessary connection management flexibility. Using either method, you can set client information on some connections and not others, as well as set different client information on different database connections from the same data source.

- Use the IBM proprietary *setClientInformation(Properties)* API. The API is defined on the *WSConnection* class, which is part of the *com.ibm.websphere.rsadapter* package.
  1. Cast the connection objects in your applications to *com.ibm.websphere.rsadapter.WSConnection* before calling the API.
  2. Optionally, set a properties object for adding new client information on a connection if and when new information is introduced by the backend database:

     ```
     public void setClientInformation (Properties props)throws SQLException;
     ```
- You can also activate the function implicitly (that is, within Application Server) by using the *WAS.clientinfo* trace string. Enable this trace dynamically, from the administrative console, just as you activate any other trace. For more information about passing client information implicitly, see “Implicitly set client information” on page 760.

See “Example: setClientInformation(Properties) API.”

Example: setClientInformation(Properties) API
Usage Scenario

This API enables you to set client information on the WebSphere Application Server connection. Some of the client information is passed on to the backend database if that database supports such functionality.

Example

```
import com.ibm.websphere.rsadapter.WSConnection;
.....
try {
   InitialContext ctx = new InitialContext();
   //Perform a naming service lookup to get the DataSource object.
   DataSource ds = (javax.sql.DataSource)ctx.lookup("java:comp/jdbc/myDS");
  }catch (Exception e) {System.out.println("got an exception during lookup: " + e);}

  WSConnection conn = (WSConnection) ds.getConnection();
  Properties props = new properties();
  props.setProperty(WSConnection.CLIENT_ID, "user123");
  props.setProperty(WSConnection.CLIENT_LOCATION, "127.0.0.1");
  props.setProperty(WSConnection.CLIENT_ACCOUNTING_INFO, "accounting");
  props.setProperty(WSConnection.CLIENT_APPLICATION_NAME, "appname");
  props.setProperty(WSConnection.CLIENT_OTHER_INFO, "cool stuff");
  conn.setClientInformation(props);
  conn.close()
```

Parameters

props contains the client information to be passed. Possible values are:

- WSConnection.CLIENT_ACCOUNTING_INFO
- WSConnection.CLIENT_LOCATION
- WSConnection.CLIENT_ID
- WSConnection.CLIENT_APPLICATION_NAME
- WSConnection.CLIENT_OTHER_INFO
- WSConnection.OTHER_CLIENT_TYPE

Refer to the WSConnection documentation for more details on which client information is passed to the backend database. To reset the client information, call the method with a null parameter.

Exceptions

This API creates an SQL exception if the database issues an exception when setting the data.

Passing client info to a db cdat_clientinfo

Implicitly set client information

You can choose to *explicitly* pass the client information of application requests to database connections by calling an IBM proprietary API, setClientInformation(Properties), on the com.ibm.websphere.rsadapter.WSConnection object within your application code. In some cases, however, you might want WebSphere Application Server to handle the passing of client information to database connections. This method of setting the client information is referred to as *implicit*. You might choose the implicit method because:

- You want to keep your application free of proprietary APIs, or
- Your application uses container-managed persistence (CMP), in which case you cannot use the proprietary API to set client information on database connections.

The WebSphere Application Server trace facility provides the capability for setting client information implicitly. You can designate one of two special trace groups to enable or disable client information passing: “WAS.clientinfo trace” or “WAS.clientinfopluslogging trace” on page 762.

Possible run-time scenarios

- Connection sharing

  In the case of connection sharing, WebSphere Application Server sets the client information on the first acquired connection handle only. If connection sharing is enabled and two or more getConnection methods are called (resulting in two handles on the same connection), only the first getConnection call causes the client information to pass to the backend database. This scenario does not apply to the explicit process of passing client information; in such cases every setClientinformation method is relayed to the database regardless of connection sharing.

- Implicit/explicit co-existence

  When you use both the explicit and implicit procedures for relaying client information, some combination of the explicitly set data and implicitly set data is combined, but the explicit setting usually takes precedence. For example, if the application sets the client accounting information to ″myAccountingInfo″, the final accountingInfo string that is passed to the backend database looks something like the following sample code:

  ```
  000325_WSRdbManagedConnectionImpl@1234_myAccountingInfo:
  ```

  where `000325` is the thread id and `WSRdbManagedConnectionImpl@1234` is the WebSphere connection instance.

- Client information reset

  When you configure Application Server to pass client information, it does reset client information when a connection is returned to the pool, but *only* if the WAS.clientinfo and WAS.clientinfopluslogging trace mechanisms are disabled (that is, WAS.clientinfo=all=disabled:WAS.clientinfopluslogging=all=disabled).

  In the explicit case, however, the reset operation is done only when the application issues setClientInformation(null) on the WSConnection connection.

WAS.clientinfo trace

By default, the implicit mechanism is disabled. You can turn on this mechanism dynamically, without stopping and starting your application server, or statically by setting the WebSphere Application Server trace group *WAS.clientinfo=all=enabled*.

The information implicitly collected and set on the database connection consists of the *user name*, *user location* and *application name*.

Important: User name and user location can only be implicitly collected and set on the database connection if you enable global security.

user name
: The name of the user that initiates the application request. This option is collected and passed to the backend database (when supported) only if global security is enabled. Information here is collected by calling the WSSecurityHelper.getFirstCaller method.

user location
: The name of the location of the user, in the form of cell:node:server. This option is collected and passed to the backend database, when appropriate, only when global security is enabled. Information here is collected by calling the WSSecurityHelper.getFirstServer method.

application name
: The name of the application running. This value is the output of the getApplication method from the J2EEName object. This value is collected regardless of the Global Security setting.

WAS.clientinfopluslogging trace

When debugging database problems, such as deadlocks, there is a set of information that is needed to help with the debugging effort. This information is typically obtained by enabling a WebSphere Relational Resource Adapter (RRA) trace, and an Enterprise JavaBean (EJB) container trace. However, there are some cases where timing is an issue when reproducing a given problem. Having too much tracing information can alter the behavior of the application, such as change the timing, and the problem might no longer occur.

Because of this situation, a new trace group is provided where only a minimum set of information is collected. This trace group is WAS.clientinfopluslogging. This function sets the client information implicitly on the connection, just like the WAS.clientinfo trace, as well as logs and traces important application activities. Those activities are:

- SQL Strings that are run (such as, select userId from tabl1 where id=? for update).
- Start, commit, and rollback of transactions.
- EJB calls (such as, Create, Remove, findByPrimaryKey, and so on).

Setting client information traces with the administrative console

Enabling the WAS.clientinfo and WAS.clientinfopluslogging traces is done through the administrative console.

1. Open the administrative console.
2. Select **Troubleshooting**.
3. Select **Log and Trace**.
4. Select the server you want to use.
5. Select **Change log detail levels**.
6. Select the **Configuration** or **Run time** tab.
7. In the **Trace Specification** entry field, type `WAS.clientinfopluslogging=all`. This starts the WAS.clientinfopluslogging trace, which passes the client information to the backend data store (if your database supports receiving client information in this manner) and logs important client activities. Type `WAS.clientinfo=all` to pass client information to the backend without tracing of client activities. To deactivate either trace, simply type the trace group name followed by `=off` (without spacing between characters).

About Cloudscape v10.1.x

WebSphere Application Server Version 6.1.x requires Cloudscape to run at a minimal version of v10.1x, which differs radically from previous releases of the database.

Use Cloudscape v10.1.x as a test and development database only.

The new code base

Unlike versions 5.1.60x and earlier, the new Cloudscape is a pure Java database server. The Cloudscape v10.1.x code base, which the open source community calls Derby, is a product of the Apache Software Foundation (ASF) open source relational database project. The new Cloudscape includes Derby without any modification to the underlying source code. Learn more about Derby code at the Apache Derby website: http://db.apache.org/derby/.

Support for two phase-commit transactions over the Network Server framework

Only the Network Server framework provides support for multiple Java virtual machines (JVMs), such as application servers, to access Cloudscape. Earlier versions of Cloudscape cannot conduct two

phase-commit transactions over the Network Server framework. However, the new Cloudscape can conduct XA transactions over Network Server; the new Derby Client JDBC driver gives Cloudscape v10.1.x that capability.

New Tools

Cloudscape v10.1.x is equipped with the following `.bat/sh` tools:
- `sysinfo`: displays database version information
- `ij`: manipulates the database instances

  transition: Use `ij` as an alternative for the old Cloudscape `cview` tool, which does not exist in version 10.1.x.
- `dblook`: dumps DDL information
- `networkServerControl`: controls the networkServer process (can be used for functions such as ping and trace)
- `startNetworkServer`: starts the networkServer process
- `stopNetworkServer`: stops the networkServer process

Note: When you run ij, surround the dbname by double quotation marks (″ ″) if it includes the full path name; for example: ij> connect ’″c:\temp;create=true″’

This is ’ ″ ″ ’ without spaces.

Continued support

The new Cloudscape package that is bundled with WebSphere Application Server shares this important feature with earlier packages: It is backed by full IBM Quality Assurance (QA).

Verifying the Cloudscape v10.1.x automatic migration

WebSphere Application Server Version 6.1.x requires Cloudscape to run at a minimal version of v10.1.x. (Note that Cloudscape v10.1.x is comprised of the Derby code base.) During the Application Server v6.1.x upgrade, the migration tool automatically upgrades the database instances that are accessed through the embedded framework by some internal components, such as the UDDI registry. The tool also attempts to upgrade Cloudscape instances that your applications access through the embedded framework. You must verify the migration results for these backend databases.

Do not use Cloudscape v10.1.x as a production database. Use it for development and test purposes only.

Learn more: The new version of Cloudscape combines the Derby runtime with additional benefits, such as IBM Quality Assurance (QA) and national language support (NLS). For information about the Cloudscape v10.1.x open source code base, see the Cloudscape section of **ibm**.com: http://www-306.ibm.com/software/data/cloudscape/.

The migration tool attempts to upgrade Cloudscape database instances that are accessed through the embedded framework only. You must manually upgrade Cloudscape instances that transact with application servers on the Network Server framework. (See the “Upgrading Cloudscape manually” on page 766 article.) This requirement eliminates the risk of corrupting third party applications that use the Network Server framework to access the same database instances as WebSphere Application Server.

Cloudscape requires the Network Server framework to transact with more than one Java Virtual Machine (JVM) concurrently. Within WebSphere Application Server, each application server is a JVM; hence you can use a single instance of Cloudscape Network Server framework with a clustered or coexistence implementation of Application Server. For more information on the Network Server framework, consult the IBM Cloudscape information center at http://publib.boulder.ibm.com/infocenter/cscv/v10r1/index.jsp.

For database instances that your applications access through the embedded framework, the automatic migration can succeed completely, fail completely, or succeed with warnings. A migration that produces warning messages does create a Cloudscape v10.1.x database with your data, but does not migrate all of your configured logic and other settings, such as:

- keys
- checks
- views
- triggers
- aliases
- stored statements

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

1. Open the post-upgrade log of each new WebSphere Application Server Version 6.1x profile. The path name of the log is `WAS_HOME/profiles/profileName/logs/WASPostUpgrade.timestamp.log`.
2. Examine the post-upgrade log for database error messages. These exceptions indicate database migration failures. The following lines are an example of post-upgrade log content, in which the database error code is `DSRA7600E`. (The migration tool references all database exceptions with the prefix `DSRA`.)

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

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

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

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

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

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

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

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

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

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

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

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

shutting down databases

shutting down databases took   0.055 seconds
```

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

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

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

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

Table 7. New class information

Class type	Old value	New value
JDBC provider class path	`${CLOUDSCAPE_JDBC_DRIVER_PATH}/db2j.jar`	`${DERBY_JDBC_DRIVER_PATH}/derby.jar` • Where `DERBY_JDBC_DRIVER_PATH` is the WebSphere environment variable that defines your Cloudscape JDBC provider • Where `derby.jar` is the base name of the JDBC driver class file (In your environment, reference the JDBC driver class file by the full path name.)
Data source implementation class: Connection pool	`com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource`	`org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource`
Data source implementation class: XA	`com.ibm.db2j.jdbc.DB2jXADataSource`	`org.apache.derby.jdbc.EmbeddedXADataSource`
Data source helper class	`com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`	`com.ibm.websphere.rsadapter.DerbyDataStoreHelper`

  Additionally, the `db2j.properties` file changes:
  - The name `WAS_HOME/cloudscape/dbj.properties` changes to `WAS_HOME/derby/derby.properties`
  - Within the file, property names change from `db2j.drda.*` to `derby.drda.*`
- A partial or a completely successful database migration changes the location and name of the database according to the following example:
  - Old database name: `c:\temp\mydb`

– New database name: WAS_HOME\profiles*profilename*\databases\c_temp_*mydb*

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

If you experience a partial migration, attempt to troubleshoot the new v10.1.x database only if you have expert knowledge of Cloudscape. Otherwise, delete the new database. Perform the manual migration procedure on the original database, just as you do for each database that completely fails auto-migration. Consult “Upgrading Cloudscape manually” for instructions.

Upgrading Cloudscape manually

During the WebSphere Application Server v6.1.x upgrade, the migration tool attempts to upgrade instances of Cloudscape that are accessed through the embedded framework only. (The new version of Cloudscape is version 10.1.x, which is based on Derby.) The automatic upgrade excludes Cloudscape instances that transact with applications through the Network Server framework. This exclusion eliminates the risk of corrupting third party applications that access the same database instances as WebSphere Application Server. You must manually upgrade database instances that are accessed through the Network Server framework. Do the same for databases that fail the automatic migration.

Do not use Cloudscape v10.1.x as a production database. Use it for development and test purposes only.

Learn more: The new version of Cloudscape combines the Derby runtime with additional benefits, such as IBM Quality Assurance (QA) and national language support (NLS).

- For information about the Cloudscape v10.1.x open source code base, see the Cloudscape section of ibm.com: http://www-306.ibm.com/software/data/cloudscape/.
- You can investigate incompatibilities between Cloudscape v10.1.x and v5.1.60x (plus versions prior to v5.1.60x) by consulting the Cloudscape migration document at http://publib.boulder.ibm.com/epubs/html/c1894711.html.

For instances of Cloudscape that are accessed through the embedded framework, determine which instances completely failed the automatic upgrade process and which ones were only partially upgraded. The article “Verifying the Cloudscape v10.1.x automatic migration” on page 763 documents how to uncover database errors and diagnostic data from various migration logs. The log messages contain the exact old and new database path names that you must use to run the manual migration. Note these new path names precisely.

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

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

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

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

1. **Network Server framework only:** Ensure that every client of the Cloudscape database can support Cloudscape v10.1.x. WebSphere Application Server clients of the database must run versions 6.1.x or 6.02.x of Application Server.

2. **Network Server framework only:** Take the database offline. No clients can access it during the migration process.
3. Examine a sample Cloudscape migration script that Application Server provides: either `db2jmigrate.bat`, or `db2jmigrate.sh`. The path of both scripts is `WAS_HOME\derby\bin\embedded\...` . You can modify the script according to the requirements of your environment. Consult the Cloudscape migration document at http://publib.boulder.ibm.com/epubs/html/c1894711.html for information about options that you can use with the script. For example, you can use the option `-DB2j.migrate.ddlFile=`*`filename`* to specify the DDL file for the new database.
4. To generate database debug logs when you run the migration script, ensure that the *debug migration trace* is active. By default, this trace function is enabled. To reactivate the debug trace if it is disabled, set one of the following trace options:
   - all traces*=all
   - com.ibm.ws.migration.WASUpgrade=all
5. Specify your old database name and the full post-migration path of the new database name when you run the script. For example: `E:\WebSphere\AppServer\derby\bin\embedded>db2jMigrate.bat` *`myOldDB myNewDB`* The logs from the automatic migration provide the exact path names to specify for both the old database and the target database. You must use this target database name to specify the new database, because your migrated Cloudscape data sources (updated by the WebSphere Application Server migration utilities) now point to the target database name. The following sample text demonstrates how log messages display target database names:

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

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

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

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

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

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

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

   The following lines are a sample of debug text.

   ```
   sourceDBURL=jdbc:db2j:E:\WebSphere\myOldDB
    newDBURL=jdbc:derby:e:\tempo\myNewDB
    ddlOnly=false
   connecting to source db <jdbc:db2j:E:\WebSphere\myOldDB>
   ```

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

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

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

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

Database performance tuning

Database performance tuning can dramatically affect the throughput of your application. For example, if your application requires high concurrency (multiple, simultaneous interactions with backend data), an improperly tuned database can result in a bottleneck. Database access threads accumulate in a backlog when the database is not configured to accept a sufficient number of incoming requests.

Because WebSphere Application Server supports the integration of many different database products, each one with unique tuning configurations, consult your database vendor documentation for comprehensive tuning information. This information center provides introductory material on DB2 tuning parameters for your convenience.

DB2 tuning parameters

DB2 has many parameters that you can configure to optimize database performance. For complete DB2 tuning information, refer to the *DB2 UDB Administration Guide: Performance* document.

DB2 logging

- **Description:** DB2 has corresponding log files for each database that provides services to administrators, including viewing database access and the number of connections. For systems with multiple hard disk drives, you can gain large performance improvements by setting the log files for each database on a different hard drive from the database files.
- **How to view or set:** At a DB2 command prompt, issue the command: `db2 update db cfg for [database_name] using newlogpath [fully_qualified_path]`.
- **Default value:** Logs reside on the same disk as the database.
- **Recommended value:** Use a separate high-speed drive, preferably performance enhanced through a redundant array of independent disk (RAID) configuration.

For more information about using AIX with DB2 see Tuning AIX systems.

DB2 configuration advisor

Located in the DB2 Control Center, this advisor calculates and displays recommended values for the DB2 buffer pool size, the database, and the database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

Number of connections to DB2 - MaxAppls and MaxAgents

When configuring the data source settings for the databases, confirm the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to establish clones, set the MaxAppls value as the maximum number of connections multiplied by the number of clones. The same relationship applies to the session manager number of connections. The MaxAppls setting must be equal to or greater than the number of connections. If you are using the same database for session and data sources, set the MaxAppls value as the sum of the number of connection settings for the session manager and the data sources.

For example, MaxAppls = (number of connections set for the data source + number of connections in the session manager) multiplied by the number of clones.

After calculating the MaxAppls settings for the WebSphere Application Server database and each of the application databases, verify that the MaxAgents setting for DB2 is equal to or greater than the sum of all of the MaxAppls values. For example, MaxAgents = sum of MaxAppls for all databases.

DB2 buffpage

- **Description:** Improves database system performance. Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. Data is accessed much faster from memory than from disk.
- **How to view or set:** To view the current value of buffpage for database *x*, issue the DB2 command `get db cfg for x` and look for the value **BUFFPAGE**. To set **BUFFPAGE** to a value of *n*, issue the DB2 command `update db cfg for x` using **BUFFPAGE *n*** and set **NPAGES** to -1 as follows:

  ```
  db2   <-- go to DB2 command mode, otherwise the following "select" does not work as is
      connect to x    <-- (where x is the particular DB2 database name)
      select * from syscat.bufferpools
         (and note the name of the default, perhaps: IBMDEFAULTBP)
         (if NPAGES is already -1, there is no need to issue following command)
      alter bufferpool IBMDEFAULTBP size -1
      (re-issue the above "select" and NPAGES now equals -1)
  ```

  You can collect a snapshot of the database while the application is running and calculate the buffer pool hit ratio as follows:

  1. Collect the snapshot:
     a. Issue the **update monitor switches using bufferpool on** command.
     b. Make sure that bufferpool monitoring is on by issuing the **get monitor switches** command.
     c. Clear the monitor counters with the **reset monitor all** command.
  2. Run the application.
  3. Issue the **get snapshot for all databases** command before all applications disconnect from the database, otherwise statistics are lost.
  4. Issue the **update monitor switches using bufferpool off** command.
  5. Calculate the hit ratio by looking at the following database snapshot statistics:
     – Buffer pool data logical reads
     – Buffer pool data physical reads
     – Buffer pool index logical reads
     – Buffer pool index physical reads
- **Default value:** 250
- **Recommended value:** Continue increasing the value until the snapshot shows a satisfactory hit rate.

The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk to service a page request. That is, the page is already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk input and output. Calculate the buffer pool hit ratio as follows:

- P = buffer pool data physical reads + buffer pool index physical reads
- L = buffer pool data logical reads + buffer pool index logical reads
- Hit ratio = (1-(P/L)) * 100%

DB2 query optimization level

- **Description:** Sets the amount of work and resources that DB2 puts into optimizing the access plan. When a database query runs in DB2, various methods are used to calculate the most efficient access plan. The range is from 0 to 9. An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.
- **How to view or set:** The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level that is specified on the **prep** and **bind** commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the dft_queryopt setting. Dynamic SQL statements use the optimization class that is specified by the current query optimization special register, which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:

  ```
  Set current query optimization = 1
  ```

  If the current query optimization register is not set, dynamic statements are bound using the default query optimization class.
- **Default value:** 5
- **Recommended value:** Set the optimization level for the needs of the application. Use high levels only when there are very complicated queries.

DB2 reorgchk

- **Description:** Obtains the current statistics for data and rebinding. Use this parameter because SQL statement performance can deteriorate after many updates, deletes or inserts.
- **How to view or set:** Use the DB2 **reorgchk update statistics on table all** command to perform the **runstats** operation on all user and system tables for the database to which you are currently connected. Rebind packages using the **bind** command. If statistics are available, issue the **db2 -v ″select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes**″ command on DB2 CLP. If no statistic updates exist, nleaf and nlevels are -1, and stats_time has an empty entry (for example: ″-″). If the runstats command was previously run, the real-time stamp from completion of the runstats operation also displays under stats_time. If you think the time shown for the previous runstats operation is too old, run the runstats command again.
- **Default value:** None
- **Recommended value:** None

DB2 locktimeout

- **Description:** Specifies the number of seconds that an application waits to obtain a lock. Setting this property helps avoid global deadlocks for applications.
- **How to view or set:** To view the current value of the lock timeout property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the value, LOCKTIMEOUT. To set LOCKTIMEOUT to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **LOCKTIMEOUT *n***, where *xxxxxx* is the name of the application database and *n* is a value between 0 and 30 000 inclusive.
- **Default value:** -1, meaning lock timeout detection is turned off. In this situation, an application waits for a lock if one is not available at the time of the request, until either of the following events occurs:
  - The lock is granted
  - A deadlock occurs

- **Recommended value:** If your database access pattern tends toward a majority of writes, set this value so that it gives you early warning when a timeout occurs. A setting of 30 seconds suits this purpose. If your pattern tends toward a majority of reads, either accept the default lock timeout value, or set the property to a value greater than 30 seconds.

DB2 maxlocks
- **Description:** Specifies the percentage of the lock list that is reached when the database manager performs escalation, from row to table, for the locks held by the application. Although the escalation process does not take much time, locking entire tables versus individual rows decreases concurrency, and potentially decreases overall database performance for subsequent attempts to access the affected tables.
- **How to view or set:** To view the current value of the maxlocks property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the MAXLOCKS value. To set MAXLOCKS to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **MAXLOCKS *n***, where *xxxxxx* is the name of the application database and *n* is a value between 1 and 100 inclusive.
- **Default value:** Refer to the current database information for property default values per operating system.
- **Recommended value:** If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the locklist parameter, which is described in the following paragraph. You can use the database system monitor to determine if lock escalations are occurring.

DB2 locklist
- **Description:** Specifies the amount of storage that is allocated to the lock list.
- **How to view or set:** To view the current value of the locklist property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the LOCKLIST value . To set LOCKLIST to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **LOCKLIST *n***, where *xxxxxx* is the name of the application database and *n* is a value between 4 and 60 000 inclusive.
- **Default value:** Refer to the current database information for property default values per operating system.
- **Recommended value:** If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the maxlocks parameter, which is described in the previous paragraph. You can use the database system monitor to determine if lock escalations are occurring. Refer to the *DB2 Administration Guide: Performance* document for more details.

Tuning parameters for data access resources

For better application performance, you can tune some data access resources through the WebSphere Application Server administrative console.

Tune these properties of data sources and connection pools to optimize the performance of transactions between your application and datastore. See the *Administering applications and their environment* PDF for more information.

Data source tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources** > *data_source* > **WebSphere Application Server connection properties**.

Enable JMS one phase optimization support

If your application does not use JMS messaging, **do not** select this option. Activating this support enables the Java Message Service (JMS) to get optimized connections from the data source. Activating this support also *prevents* JDBC applications from obtaining connections from the data source. For further explanation of JMS one phase support, refer to the article entitled ″Sharing connections to benefit from one phase commit optimization″ in this information center.

Statement cache size

Specifies the number of statements that can be cached per connection.

The WebSphere Application Server data source optimizes the processing of *prepared statements* and *callable statements* by caching those statements that are not being used in an active connection. Both statement types help reduce overhead for transactions with backend data.

- A prepared statement is a precompiled SQL statement that is stored in a `PreparedStatement` object. Application Server uses this object to run the SQL statement multiple times, as required by your application run time, with values that are determined by the run time.
- A callable statement is an SQL statement that contains a call to a stored procedure, which is a series of precompiled statements that perform a task and return a result. The statement is stored in the `CallableStatement` object. Application Server uses this object to run a stored procedure multiple times, as required by your application run time, with values that are determined by the run time.

In general, the more statements your application has, the larger the cache should be. **Be aware**, however, that specifying a larger statement cache size than needed wastes application memory and *does not* improve performance.

Determine the value for your cache size by adding the number of uniquely prepared statements and callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible statements that can be cached on a given connection over the life of the server. See the *Administering applications and their environment* PDF for more information.

Default: For most databases the default is 10. Zero means there is no cache statement.

Connection pool tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources** > *data_source* > **Connection pool settings**.

Maximum connections

Specifies the maximum number of physical connections that can be created in this pool. These are the physical connections to the backend datastore. When this number is reached, no new physical connections are created; requestors must wait until a physical connection that is currently in use is returned to the pool.

For optimal performance, set the value for the connection pool lower than the value for the Web container threadpool size. Lower settings, such as 10 to 30 connections, might perform better than higher settings, such as 100. See the *Administering applications and their environment* PDF for more information.

Default: 10

Minimum connections

Specifies the minimum number of physical connections to maintain. Until this number is exceeded, the pool maintenance thread does not discard physical connections.

If you set this property for a higher number of connections than your application ultimately uses at run time, you do not waste application resources. WebSphere Application Server does not create additional connections to achieve your minimum setting. Of course, if your application requires more connections than the value you set for this property, application performance diminishes as connection requests wait for fulfillment. See the *Administering applications and their environment* PDF for more information.

Default: 1

Managing resources through JCA lifecycle management operations

You can manage the run-time status of your data source and connection factory resources to perform some data access administrative tasks without restarting the application server. This topic outlines the process for managing those resources through the administrative console.

When you manage the run-time status of connection factories or data sources, you are applying J2EE Connector Architecture (JCA) lifecycle management operations to the MBeans that are associated with these resources. The management operations are PAUSE and RESUME. Pausing an MBean halts outbound communication to the backend, such as a database. This action affects all applications that use the corresponding connection factory or data source on the same server.

With these management operations, you can perform some administrative tasks dynamically, without restarting your application server:

- Respond to a security threat, and prevent new connection requests from reaching the backend
- Perform maintenance on the backend
- Apply configuration changes to non-required properties of the connection factory or data source, such as turning JDBC trace on or off, or modifying preferences for collecting client information

1. Navigate to the administrative console page that corresponds to the resource type that you want to manage.
   - For connection factories, use either of the following paths:
     - **Resources** > **Resource Adapters** > **J2C connection factories**
     - **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories**
   - For data sources, use either of the following paths:
     - **Resources** > **JDBC** > **Data sources**
     - **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources**
2. Select the connection factory or data source configurations that you want to manage, and click **Manage state**. The administrative console now displays the JCA lifecycle management page, which contains a table that depicts the full scope configuration of your previous selection.

   The table is comprised of three columns:
   - **JNDI name:** The Java Naming and Directory Interface (JNDI) name of the connection factory or data source configuration.
   - **Running object scope:** The server that is running the connection factory or data source MBean.
   - **Status:** The status of the connection factory or data source MBean.
3. Select the rows that represent each invocation of the resource, per running server, that you want to manage. Be aware that when you click your management operation, WebSphere Application Server applies it to every resource object in your selection.

   Restriction: If the MBean status of a row has a value of NOT_ACCESSED, you cannot apply JCA lifecycle management operations to that MBean. The NOT_ACCESSED state indicates that the MBean exists on the specified server, but no applications performed a JNDI namespace lookup on the corresponding connection factory or data source.
4. Click **Pause** or **Resume**. The status column of the table changes to reflect the new state of the MBean.

JCA lifecycle management

Use this page to perform JCA lifecycle management operations on data source and connection factory MBeans. With these management operations, you can control the runtime status of the corresponding data source and connection factory resources.

You can view this administrative console page in different locations, depending on whether you want to manage data sources or J2C connection factories. For example:

- For connection factories: Click **Resources** > **Resource adapters** > **J2C connection factories**. Select the connection factory configurations that you want to manage, and click **Manage state**.
- For data sources: Click **Resources** > **JDBC** > **Data sources**. Select the data source configurations that you want to manage, and click **Manage state**.

Guidelines for using this administrative console page:

- The table displays a list of Version 6.0.2-compatible MBeans that correspond to the data sources or connection factories in your selection from the previous console page.
- You can only perform JCA lifecycle management actions on MBeans that are in the active state. In this context, an MBean is considered active when an application performs a Java Naming and Directory Interface (JNDI) name space lookup on the corresponding data source or connection factory resource.
- Pausing an MBean halts outbound communication to the backend, such as a database. This action affects all applications that use the resource on the selected server.

Table column heading descriptions:

Name (JNDI name):

The name of the connection factory or data source configuration, followed by the Java Naming and Directory Interface (JNDI) name in parenthesis.

Running object scope:

The server that is running the connection factory or data source MBean.

Status:

The state of the connection factory or data source MBean.

Possible values:

State	Indications
ACTIVE	• The resource that corresponds with the MBean is ready to provide an application with connections to a backend. • An application performed a JNDI namespace lookup on this resource. You can apply the JCA lifecycle management operation of PAUSE to an MBean in this state.
PAUSED	• All outbound communication to the backend through the corresponding resource is stopped, as a result of a JCA lifecycle management operation that was applied previously to the MBean. • An application performed a JNDI namespace lookup on the resource. You can apply the JCA lifecycle management operation of RESUME to an MBean in this state.

State	Indications
NOT_ACCESSED	• The MBean exists on the specified server, but no applications performed a JNDI name space lookup on the corresponding connection factory or data source. You **cannot** apply JCA lifecycle management operations to an MBean in this state.

Cannot access a data source

WebSphere Application Server diagnostic tools provide services to help troubleshoot database connection problems. Additionally, the IBM Web site provides flexible searching capabilities for finding documented solutions to database-specific connection problems.

The following steps help you quickly isolate connectivity problems.

1. Browse the log files of the application server for clues.

   See Viewing JVM logs. By default, these files are *app_server_root*/*server_name*/SystemErr.log and SystemOut.log.
2. Browse the Helper Class property of the data source to verify that it is correct and that it is on the WebSphere Application Server class path. Mysterious errors or behavior might result from a missing or misnamed Helper Class name. If WebSphere Application Server cannot load the specified class, it uses a default helper class that might not function correctly with your database manager.
3. Verify that the Java Naming and Directory Interface (JNDI) name of the data source matches the name used by the client attempting to access it. If error messages indicate that the problem might be naming-related, such as referring to the **name server** or **naming service**, or including error IDs beginning with **NMSV**, look at the Naming related problems and Troubleshooting the naming service component topics.
4. Enable tracing for the resource adapter using the trace specification, RRA=all=enabled. Follow the instructions for dumping and browsing the trace output, to narrow the origin of the problem.

For a comprehensive list of database-specific troubleshooting tips, see the WebSphere support page at http://www-306.ibm.com/software/webservers/appserv/was/support/. In the Search Support field, type a database vendor name among your search terms. Select **Solve a problem**, then click **Search**.

Remember that you can always find Support references in the Troubleshooting help from IBM article of this information center.

Currently this information center provides a limited number of troubleshooting tips for the following databases:

- Oracle
- DB2
- SQL Server
- Cloudscape
- Sybase
- General data access problems

General data access problems

- An exception ″IllegalConnectionUseException″ occurs
- WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred.
- ConnectionWaitTimeoutException.
- com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name ″NULL″ could not be found. SQLSTATE=42705
- java.sql.SQLException: java.lang.UnsatisfiedLinkError:

- ″J2CA0030E: Method enlist caught java.lang.IllegalStateException″ wrapped in error ″WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred″ when attempting to execute a transaction.
- java.lang.UnsatisfiedLinkError:xaConnect exception when attempting a database operation
- ″J2CA0114W: No container-managed authentication alias found for connection factory or datasource *datasource*″ when attempting a database operation
- An error is thrown if you use the ws_ant command to perform the database customization for Structured Query Language in Java on HP platforms
- Container-managed persistence (CMP) cannot successfully obtain the database access function as defined.

IllegalConnectionUseException

This error can occur because a connection obtained from a WAS40DataSource is being used on more than one thread. This usage violates the J2EE 1.3 programming model, and an exception generates when it is detected on the server. This problem occurs for users accessing a data source through servlets or bean-managed persistence (BMP) enterprise beans.

To confirm this problem, examine the code for connection sharing. Code can inadvertently cause sharing by not following the programming model recommendations, for example by storing a connection in an instance variable in a servlet, which can cause use of the connection on multiple threads at the same time.

WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred

This error can occur because:
- An attempt was made to share a single-phase connection, when each **getConnection** method has different connection properties; such as the AccessIntent. This attempt causes a non-shareable connection to be created.
- An attempt was made to have more than one unshareable connection participate in a global transaction, when the data source is not an XA resource.
- An attempt was made to have a one-phase resource participate in a global transaction while an XA resource or another one-phase resource already participated in this global transaction.
  - Within the scope of a global transaction you try to get a connection more than once and at least one of the resource-refs you use specifies that the connection is unshareable, and the data source is not configured to support two-phase commit transactions. It does not support an XAResource. If you do not use a resource-ref, you default to unshareable connections.
  - Within the scope of a global transaction you try to get a connection more than once and at least one of the resource-refs you use specifies that the connection is shareable and the data source is not configured to support two-phase commit transactions. That is, it does not support an XAResource. In addition, even though you specify that connections are shareable, each getConnection request is made with different connection properties (such as IsolationLevel or AccessIntent). In this case, the connections are not shareable, and multiple connections are handed back.
  - Multiple components (servlets, session beans, BMP entity beans, or CMP entity beans) are accessed within a global transaction. All use the same data source, all specify shareable connections on their resource-refs, and you expect them to all share the same connection. If the properties are different, you get multiple connections. AccessIntent settings on CMP beans change their properties. To share a connection, the AccessIntent setting must be the same. For more information about CMP beans sharing a connection with non-CMP components, see the *Data access application programming interface support* and *Example: Accessing data using IBM extended APIs to share connections between container-managed and bean-managed persistence beans* topics in the DataAccess section of the information center.

To correct this error:
- Check what your client code passes in with its getConnection requests, to ensure they are consistent with each other.

- Check the connection sharing scope from the resource binding, using an assembly tool.
  - If you are running an unshareable connection scope, verify that your data source is an XA data source.
  - If you are running a shareable connection scope, verify that all connection properties, including AccessIntent, are sharable.
- Check the JDBC provider implementation class from the Manage JDBC resource panel of the administrative console to ensure that it is a class that supports XA-type transactions.

ConnectionWaitTimeoutException accessing a data source or resource adapter

If your application receives exceptions like a com.ibm.websphere.ce.cm.ConnectionWaitTimeoutException or com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException when attempting to access a WebSphere Application Server data source or JCA-compliant resource adapter, respectively, some possible causes are:

- The maximum number of connections for a given pool is set too low. The demand for concurrent use of connections is greater then the configured maximum value for the connection pool. One indication that this situation is the problem is that you receive these exceptions regularly, but your CPU utilization is not high. This exception indicates that there are too few connections available to keep the threads in the server busy.
- Connection Wait Time is set too low. Current demand for connections is high enough such that sometimes there is not an available connection for short periods of time. If your connection wait timeout value is too low, you might timeout shortly before a user returns a connection back to the pool. Adjusting the connection wait time can give you some relief. One indication of this problem is that you use close to the maximum number of connections for an extended period and receiving this error regularly.
- You are not closing some connections or you are returning connections back to the pool at a very slow rate. This situation can happen when using unshareable connections, when you forget to close them, or you close them long after you are finished using them, keeping the connection from returning to the pool for reuse. The pool soon becomes empty and all applications get ConnectionWaitTimeoutExceptions. One indication of this problem is you run out of connections in the connection pool and you receive this error on most requests.
- You are driving more load than the server or backend system have resources to handle. In this case you must determine which resources you need more of and upgrade configurations or hardware to address the need. One indication of this problem is that the application or database server CPU is nearly 100% busy.

To correct these problems, either:

- Modify an application to use fewer connections
- Properly close the connections.
- Change the pool settings of MaxConnections or ConnnectionWaitTimeout.
- Adjust resources and their configurations.

com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name ″NULL″ could not be found. SQLSTATE=42705

This error occurs when a data source is defined but the **databaseName** attribute and the corresponding value are not added to the custom properties panel.

To add the **databaseName** property:

1. Click **Resources>Manage JDBC Providers** link in the administrative console.
2. Select the JDBC provider that supports the problem data source.
3. Select **Data Sources** and then select the problem data source.
4. Under **Additional properties** click **Custom Properties**.

5. Select the **databaseName** property, or add one if it does not exist, and enter the actual database name as the value.
6. Click **Apply** or **OK**, and then click **Save** from the action bar.
7. Access the data source again.

java.sql.SQLException: java.lang.UnsatisfiedLinkError:

This error indicates that the directory containing the binary libraries which support a database are not included in the LIBPATH environment variable for the environment in which the WebSphere Application Server starts.

The path containing the DBM vendor libraries vary by dbm. One way to find them is by scanning for the missing library specified in the error message. Then you can correct the LIBPATH variable to include the missing directory, either in the `.profile` of the account from which WebSphere Application Server is executed, or by adding a statement in a `.sh` file which then executes the `startServer` program.

″J2CA0030E: Method enlist caught java.lang.IllegalStateException″ wrapped in error ″WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred″ when attempting to execute a transaction.

This error can occur when last participant support is missing or disabled. last participant support allows a one-phase capable resource and a two-phase capable resource to enlist within the same transaction.

Last participant support is only available if the following are true:
- WebSphere Application Server Programming Model Extensions (PME) is installed. PME is included in the Application Server Integration Server product.
- The Additional Integration Server Extensions option is enabled when PME is installed. If you perform a typical installation, this function is enabled by default. If you perform a custom installation, you have the option to disable this function, which disables last participant support.
- The application enlisting the one-phase resource is deployed with the **Accept heuristic hazard** option enabled. This deployment is done with an assembly tool such as the Application Server Toolkit (AST) or Rational Web Developer.

java.lang.UnsatisfiedLinkError:xaConnect exception when attempting a database operation

This problem has two main causes:
- The most common cause is that the jdbc driver which supports connectivity to the database is missing, or is not the correct version, or that native libraries which support the driver are on the system's path.
  - To resolve this problem on a Windows platform, verify that the JDBC driver jar file is on the system PATH environment variable:
    - If you are using DB2,verify that at least the DB2 client product has been installed on the WebSphere host
      - On DB2 version 7.2 or earlier, the file where the client product is installed on the WebSphere Application Server is db2java.zip. Verify that the usejdbc2.bat program has been executed after the database install and after any upgrade to the database product.
      - On DB2 version 8.1 or later, use the DB2 Universal JDBC Provider Driver when defining a JDBC provider in WebSphere Application Server. The driver file is db2jcc.jar. If you use the type 2 (default) option, verify that at least the DB2 client product is installed on the WebSphere Application Server host. If you specify the type 4 option, the DB2 client does not need to be installed, but the file db2jcc.jar still must be present.

        When specifying the location of the driver file, it is recommended to that you specify the path and file name of the target DB2 installation, rather than simply copying the file to a local

directory, if possible. Otherwise, you may be exposed to problems if the target DB2 installation is upgraded and the driver used by WebSphere Application Server is not. If you choose **DB2 Legacy CLI-based type 2 JDBC Driver** when defining a JDBC provider in WebSphere Application Server, then you must still follow the steps to ensure that you have the correct version of the db2java.zip file (see instructions for DB2 7.2 or earlier).

– On operating systems such as AIX or Linux, ensure that any native libraries required to support the database client of your database product are specified in the LD_LIBRARY_PATH environment variable in the profile of the account under which WebSphere Application Server executes.

  If you are using DB2 The native library is libdb2jdbc.so. The best way to ensure that this library is accessed correctly by WebSphere is to call the db2profile script supplied with DB2 from the .profile script of the account (such as ″root″) under which WebSphere runs.

  - If you are using DB2 version 7.2 or earlier, ensure that the usejdbc2,script provided with DB2 is called from the profile of the account under which WebSphere Application server is launched.
  - If you are using DB2 version 8.1 or later, see the previous instructions for the Windows operating system.

• If the database manager is DB2, you may have chosen the option to create a 64-bit instance. Sometimes a 64-bit configuration is not supported. If this has happened, remove the database instance and create a new one with the default 32-bit setting.

  If you are using a CLI driver or a Universal JDBC T2 driver, WebSphere Application Server does support interaction with a DB2 UDB 64-bit server, but it must be through a DB2 UDB 32-bit client. The WebSphere Application Server environment (CLASSPATH and so on) must use the 32-bit client code to ensure correct function.

  With a Universal JDBC T4 driver, you do not need the 32-bit DB2 client. You need only configure the CLASSPATH to include db2jcc.jar and its license files in the WebSphere Application Server environment.

  Note: For general help in configuring JDBC drivers and data sources in WebSphere Application Server, see the topic Accessing data from applications.

″J2CA0114W: No container-managed authentication alias found for connection factory or datasource *datasource*″ when attempting a database operation

This error might occur in the SystemOut.log file when you run an application to access a data source after creating the data source using JACL script.

The error message occurs because the JACL script did not set container-managed authentication alias for CMP connection factory. The JACL is missing the following line:

```
$AdminConfig create MappingModule $cmpConnectorFactory  "{mappingConfigAlias
DefaultPrincipalMapping} {authDataAlias $authDataAlias}
```

To correct this problem, add the missing line to the JACL script and run the script again. See “Example: Creating a JDBC provider and data source using Java Management Extensions API and the scripting tool” on page 746 for a sample JACL script.

An error is thrown if you use the ws_ant command to perform the database customization for Structured Query Language in Java on HP platforms

If you use the `ws_ant` command to perform the database customization for Structured Query Language in Java (SQLJ) on HP platforms, you can receive an error similar to the following:

```
[java] [ibm][db2][jcc][sqlj]
[java] [ibm][db2][jcc][sqlj] Begin Customization
[java] [ibm][db2][jcc][sqlj] encoding not supported!!
```

The cause of this error might be that your databases were created using the HP default character set. The Java Common Client (JCC) driver depends on the software development kit (SDK) to perform the codepage conversions. The SDK shipped with this product, however, does not support the HP default codepage.

You need to set your LANG to the ISO locale before creating the databases. It should be similar to the following:

```
export LANG=en_US.iso88591
```

Refer to the DB2 Tech Notes at `http://www-3.ibm.com/software/data/db2/udb/ad/v8/bldg/t0004877.htm` for details.

Container-managed persistence (CMP) cannot successfully obtain the database access function as defined.

When WebSphere Application Server is caching certain generated code that is accessed in the database on the connection factory, and if any changes in the Java archive (JAR) file require regeneration of the database access, the changes are not effective until you stop and restart the server.

Examples of when this failure might occur include:

- Adding an enterprise bean custom finder method; a NullPointerException exception is created.
- Updating an enterprise bean custom finder method; the new SQL statement does not run.
- Changing schema mapping; the new SQL statement does not run.

In summary, if you add or update an enterprise bean that contains a custom finder method, you must stop and then restart the server.

Problems accessing an Oracle data source

This article provides troubleshooting tips for accessing Oracle data sources.

What kind of error do you see when you try to access your Oracle-based data source?

- "An invalid Oracle URL is specified"
- ""DSRA0080E: An exception was received by the data store adapter. See original exception message: ORA-00600″ when connecting to or using an Oracle data source" on page 781
- "DSRA8100E: Unable to get a {0} from the DataSource. Explanation: See the linkedException for more information." on page 781
- ""Error while trying to retrieve text for error″ error when connecting to an Oracle data source" on page 781
- ""java.lang.UnsatisfiedLinkError:″ connecting to an Oracle data source" on page 781
- "java.lang.NullPointerException referencing 8i classes, or ″ internal error: oracle.jdbc.oci8. OCIEnv″ connecting to an Oracle data source" on page 782
- "WSVR0016W: Classpath entry for the Oracle JDBC Thin Driver has an invalid variable" on page 782
- "Transaction recovery failure (for XA data sources)" on page 782

An invalid Oracle URL is specified

This error might be caused by an incorrectly specified URL on the URL property of the target data source.

Examine the URL property for the data source object in the administrative console. For the 8i OCI driver, verify that **oci8** is used in the URL. For the 9i OCI driver, you can use either **oci8** or **oci**.

Examples of Oracle URLs:

- For the thin driver: `jdbc:oracle:thin:@hostname.rchland.ibm.com:1521:IBM`
- For the thick (OCI) driver: `jdbc:oracle:oci8:@tnsname1`

″DSRA0080E: An exception was received by the data store adapter. See original exception message: ORA-00600″ when connecting to or using an Oracle data source

A possible reason for this exception is that the version of the Oracle JDBC driver being used is older than the Oracle database. It is possible that more than one version of the Oracle JDBC driver is configured on the WebSphere Application Server.

Examine the version of the JDBC driver. Sometimes you can determine the version by looking at the class path to determine what directory the driver is in.

If you cannot determine the version this way, use the following program to determine the version. Before running the program, set the class path to the location of your JDBC driver files.

```
import java.sql.*;
     import oracle.jdbc.driver.*;
     class JDBCVersion
     {
         public static void main (String args[])
         throws SQLException
         {
             // Load the Oracle JDBC driver
             DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
             // Get a connection to a database
             Connection conn = DriverManager.getConnection
             ("jdbc:oracle:thin:@appaloosa:1521:app1","sys","change_on_install");
             // Create Oracle DatabaseMetaData object
             DatabaseMetaData meta = conn.getMetaData();
             // gets driver info:
             System.out.println("JDBC driver version is " + meta.getDriverVersion());
         }
     }
```

If the driver and the database are at different versions, replace the JDBC driver with the correct version. If multiple drivers are configured, remove any that occur at the incorrect level.

DSRA8100E: Unable to get a {0} from the DataSource. Explanation: See the linkedException for more information.

When using an oracle thin driver, Oracle creates a ″java.sql.SQLException: invalid arguments in call″ error if no user name or password is specified when getting a connection. If you see this error while running WebSphere Application Server, the alias is not set.

To remove the exception, define the alias on the data source.

″Error while trying to retrieve text for error″ error when connecting to an Oracle data source

The most likely cause of this error is that the Oracle 8i OCI driver is being used with an ORACLE_HOME property that is either not set or is set incorrectly.

To correct the error, examine the user profile that WebSphere Application Server is running under to verify that the $ORACLE_HOME environment variable is set correctly.

″java.lang.UnsatisfiedLinkError:″ connecting to an Oracle data source

The environment variable LIBPATH might not be set or is set incorrectly, if your data source creates an **UnsatisfiedLinkError** error, and the full exception indicates that the problem is related to an Oracle module, as in the following examples.

Example of invalid an LIBPATH for the 8i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
/usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so:
load ENOENT on shared library(s)
/usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so libclntsh.a
```

Example of an invalid LIBPATH for the 9i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
no ocijdbc9  (libocijdbc9.a or .so) in java.library.path
at java.lang.ClassLoader.loadLibrary(ClassLoader.java(Compiled Code))
at java.lang.Runtime.loadLibrary0(Runtime.java:780)
```

To correct the problem, examine the user profile under which the WebSphere Application Server is running to verify that the LIBPATH environment variable includes Oracle libraries. Scan for the `lobocijdbc8.so` file to find the right directory.

java.lang.NullPointerException referencing 8i classes, or ″ internal error: oracle.jdbc.oci8. OCIEnv″ connecting to an Oracle data source

The problem might be that the 9i OCI driver is being used on an AIX 32-bit machine, the LIBPATH is set correctly, but the ORACLE_HOME environment variable is not set or is set incorrectly. You can encounter an exception similar to either of the following when your application attempts to connect to an Oracle data source:

Exception example for the java.lang.NullPointerException:

```
Exception in thread "main" java.lang.NullPointerException
at oracle.jdbc.oci8.OCIDBAccess.check_error(OCIDBAccess.java:1743)
at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:69)
at oracle.jdbc.oci8.OCIDBAccess.logon(OCIDBAccess.java:452)
at oracle.jdbc.driver.OracleConnection. <init>(OracleConnection.java:287)
```

Exception example for the java.sql.SQLException:

```
Exception in thread "main" java.sql.SQLException:
internal error: oracle.jdbc.oci8. OCIEnv@568b1d21
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:184)
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:226)
at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:79)
```

To correct the problem, examine the user profile that WebSphere Application Server is running under to verify that it has the `$ORACLE_HOME` environment variable set correctly, and that the `$LIBPATH` includes `$ORACLE_HOME/lib`.

WSVR0016W: Classpath entry for the Oracle JDBC Thin Driver has an invalid variable

This error occurs when there is no environment variable defined for the property, ORACLE_JDBC_DRIVER_PATH.

Verify this problem in the administrative console. Go to **Environment > Manage WebSphere Variables** to verify whether the variable `ORACLE_JDBC_DRIVER_PATH` is defined.

To correct the problem, click **New** and define the variable. For example, name : **ORACLE_JDBC_DRIVER_PATH** , value : **c:\oracle\jdbc\lib**. Use a value that names the directory in your operating system that contains the `ojdbc14.jar` file (or the `ojdbc14_g.jar` file to enable Oracle trace).

Transaction recovery failure (for XA data sources)

Problem

When WebSphere Application Server attempts to recover Oracle database transactions, the transaction service issues following exception:

```
WTRN0037W: The transaction service encountered an error on an xa_recover operation.
The resource was com.ibm.ws.rsadapter.spi.WSRdbXaResourceImpl@1114a62.
The error code was XAER_RMERR. The exception stack trace follows:
javax.transaction.xa.XAException
at oracle.jdbc.xa.OracleXAResource.recover(OracleXAResource.java:726)
at com.ibm.ws.rsadapter.spi.WSRdbXaResourceImpl.recover(WSRdbXaResourceImpl.java:954)
at com.ibm.ws.Transaction.JTA.XARminst.recover(XARminst.java:137)
at com.ibm.ws.Transaction.JTA.XARecoveryData.recover(XARecoveryData.java:609)
at com.ibm.ws.Transaction.JTA.PartnerLogTable.recover(PartnerLogTable.java:511)
at com.ibm.ws.Transaction.JTA.RecoveryManager.resync(RecoveryManager.java:1784)
at com.ibm.ws.Transaction.JTA.RecoveryManager.run(RecoveryManager.java:2241)
```

Cause

Oracle requires services such as the WebSphere Application Server transaction service to have special permissions for performing transaction recoveries.

Solution

As user **SYS**, run the following commands on your Oracle server:

```
grant select on pending_trans$ to public;
grant select on dba_2pc_pending to public;
grant select on dba_pending_transactions to public;
grant execute on dbms_system to < user>;
```

This problem is mentioned under Oracle bug: 3979190. Running the preceding commands solves the problem.

Problems accessing a DB2 database

This article provides troubleshooting tips for accessing DB2 databases.

What kind of problem are you having accessing your DB2 database?

- “Receive "'SYSTEM’ is not a valid authorization ID" message when trying to access DB2 on a Windows machine where WebSphere Application Server is also installed.” on page 789
- “XAException: XAER_NOTA on XA prepare call in DB2 Universal JDBC Driver type 4 after one phase transaction rollback” on page 789
- “java.rmi.MarshalException logged for application client due to incompatibility of JDBC driver file versions” on page 790
- “Database failure triggers problematic -99999 exception for applications that use DB2 Universal Driver type 4” on page 790
- “Cannot access DB2 on Linux when using the DB2 Universal JDBC Driver” on page 791
- “Test connection operation fails for DB2 legacy CLI-based Type 2 driver that is used to access DB2 running on 64-bit Linux” on page 792
- “Illegal conversion occurs on any VARCHAR FOR BIT DATA column in a container-managed persistent bean” on page 792
- Check the IBM support page for information on problems that occur when using connection pooling with DB2.

SQL0567N ″DB2ADMIN ″ is not a valid authorization ID. SQLSTATE=42602

If you encounter this error when attempting to access a DB2 Universal Database (UDB):
1. Verify that your user name and password in the data source properties page in the administrative console are correct.
2. Ensure that the user ID and password do not contain blank characters before, in between, or after.

SQL0805N Package *package-name* was not found

Possible reasons for these exceptions:
- If the package name is `NULLID.SQLLC300`, see SQL0805N Package ″NULLID.SQLLC300″ was not found. SQLSTATE=51002. for the reason.
- You are attempting to use an XA-enabled JDBC driver on a DB2 database that is not XA-ready.

To correct the problem on a DB2 Universal Database (UDB), run this one-time procedure, using the db2cmd interface while connected to the database in question:
1. **DB2 bind @db2ubind.lst blocking all grant public**
2. **DB2 bind @db2cli.lst blocking all grant public**

The `db2ubind.lst` and `db2cli.lst` files are in the `bnd` directory of your DB2 installation root. Run the commands from that directory.

SQL0805N Package ″NULLID.SQLLC300″ was not found. SQLSTATE=51002

This error can occur because:
- The underlying database was dropped and recreated.
- DB2 was ugpraded and its packages are not rebound correctly.

To resolve this problem, rebind the DB2 packages by running the db2cli.lst script found in the bnd directory. For example: db2>@db2cli.lst.

SQL30082N Attempt to establish connection failed with security reason ″17″ (″UNSUPPORTED FUNCTION’) SQLSTATE=08001

This error can occur when the security mechanism specified by the client is not valid for this server. Some typical examples:
- The client sent a new password value to a server that does not support the change password function.
- The client sent SERVER_ENCRYPT authentication information to a server that does not support password encryption.

- The client sent a userid, but no password, to a server that does not support authentication by userid only.
- The client has not specified an authentication type, and the server has not responded with a supported type. This can include the server returning multiple types from which the client is unable to choose.

To resolve this problem, ensure that your client and server use the same security mechanism. For example, if this is an error on your data source, verify that you have assigned a user id and password or authentication alias.

SQLException, with ErrorCode -99,999 and SQLState 58004, with Java ″StaleConnectionException: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004″, when using WAS40-type data source

An unexpected system failure usually occurs when running in XA mode (two-phase commit). Among the many possible causes are:
- An invalid username or password was provided.
- The database name is incorrect.
- Some DB2 packages are corrupted.

To determine whether you have a user name or password problem, look in the `db2diag.log` file to view the actual error message and SQL code. A message like the following example, with an SQLCODE of -1403, indicates an invalid user ID or password:

```
2002-07-26-14.19.32.762905   Instance:db2inst1   Node:000
PID:9086(java)   Appid:*LOCAL.db2inst1.020726191932
XA DTP Support  sqlxa_open   Probe:101
DIA4701E Database "POLICY2" could not be opened
for distributed transaction processing.
String Title: XA Interface SQLCA  PID:9086 Node:000
SQLCODE = -1403
```

To resolve these problems:
1. Correct your user name and password. If you specify your password on the GUI for the data source, ensure that the user name and password you specify on the bean are correct. The user name and password you specify on the bean overwrite whatever you specify when creating the data source.
2. Use the correct database name.
3. Rebind the packages (in the `bnd` directory) as follows:

   ```
   db2connect to dbname
   c:\SQLLIB\bnd>DB2 bind @db2ubind.lst blocking all grant public
   c:\SQLLIB\bnd>DB2 bind @db2cli.lst blocking all grant public
   ```
4. Ensure that the \WebSphere\AppServer\properties\wsj2cdpm.properties file has the right user ID and password.

Error message `java.lang.reflect.InvocationTargetException: com.ibm.ws.exception.WsException: DSRA0023E: The DataSource implementation class ″COM.ibm.db2.jdbc.DB2XADataSource″ could not be found.` when trying to access a DB2 database

One possible reason for this exception is that a user is attempting to use a JDBC 2.0 DataSource, but DB2 is not JDBC 2.0-enabled. This situation frequently happens with new installations of DB2 because DB2 provides separate drivers for JDBC 1.X and 2.0, with the same physical file name. By default, the JDBC 1.X driver is on the class path.

To confirm this problem:
- On Windows systems, look for the inuse file in the java12 directory in your DB2 installation root. If the file missing, you are using the JDBC 1.x driver.
- On operating systems such as AIX or Linux, check the class path for your data source. If the class path does not point to the `db2java.zip` file in the `java12` directory, you are using the JDBC 1.x driver.

To correct this problem:
- On Windows systems, stop DB2. Run the usejdbc2.bat file from the java12 directory in your DB2 instalation root. Run this file from a command line to verify that it completes successfully.
- On operating systems such as AIX or Linux, change the class path for your data source to point to the `db2java.zip` file in the `java12` directory of your DB2 installation root.

CLI0119E System error. SQLSTATE=58004 - DSRA8100 : Unable to get a XAconnection or DSRA0011E: Exception: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=5800

If you encounter this error when attempting to access a DB2 Universal Database (UDB) data source:
1. On the data source properties page in the administrative console, verify that the correct database name is specified on the data source.
2. On the custom properties page, check your user name and password custom properties. Verify that they are correct.
3. Ensure the user ID and password do not contain any blank characters, before, in between, or after.
4. Check that the `WAS.policy` file exists for the application, for example, `D:\WebSphere\AppServer\installedApps\markSection.ear\META-INF\was.policy`.
5. View the entire exception listing for an underlying SQL error, and look it up using the DBM vendor message reference.

If you encounter this error while running DB2 on Red Hat Linux, the **max queues system wide** parameter is too low to support DB2 while it acquires the necessary resources to complete the transaction. When this problem exists, the exceptions J2CA0046E and DSRA0010E can precede the exception DSRA8100E.

To correct this problem, edit the /proc/sys/kernal/msgmni file to increase the value of the **max queues system wide** parameter to a value greater than 128.

COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code ″2″. SQLSTATE=40001

This problem is probably an application-caused DB2 deadlock, particularly if you see an error similar to the following when accessing a DB2 data source:

```
ERROR CODE: -911
COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N
The current transaction has been rolled back because of a deadlock or timeout.
Reason code "2".  SQLSTATE=40001
```

To diagnose the problem:
1. Execute these DB2 commands:
   a. `db2 update monitor switches using LOCK ON`
   b. `db2 get snapshot for LOCKS on dbName >`

   The *directory_name*`\lock_snapshot.log` now has the DB2 lock information.
2. Turn off the lock monitor by running: `db2 update monitor switches using LOCK OFF`

To verify that you have a deadlock:
1. Look for an application handle that has a lock-wait status, and then look for the ID of the agent holding lock to verify the ID of the agent.
2. Go to that handle to verify it has a lock-wait status, and the ID of the agent holding the lock for it. If it is the same agent ID as the previous one, then you know that you have a circular lock (deadlock).

To resolve the problem:
1. Examine your application and use a less restrictive isolation level if no concurrency access is needed.
2. Use caution when changing the **accessIntent** value to move to a lower isolation level. This change can result in data integrity problems.

3. For DB2/UDB Version 7.2 and earlier releases, you can set the DB2_RR_TO_RS flag from the DB2 command line window to eliminate unnecessary deadlocks, such as when the accessIntent defined on the bean method is too restrictive, for example, PessimisticUpdate. The DB@_RR_TO_RS setting has two impacts:
   - If RR is your chosen isolation level, it is effectively downgraded to RS.
   - If you choose another isolation level, and the DB2_RR_TO_RS setting is on, a scan skips over rows that are deleted but not committed, even though the row might qualify for the scan. The skipping behavior affects the RR, Read Stability (RS), and Cursor Stability (CS) isolation levels.

   For example, consider the scenario where transaction A deletes the row with column1=10 and transaction B does a scan where column1>8 and column1<12. With DB2_RR_TO_RS off, transaction B waits for transaction A to commit or rollback. If transaction A rolls back, the row with column1=10 is included in the result set of the transaction B query. With DB2_RR_TO_RS on, transaction B does not wait for transaction A to commit or rollback. Transaction B immediately receives query results that do not include the deleted row. Setting DB2_RR_TO_RS effectively changes locking behavior, thus avoiding deadlocks.

″COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource″ could not be found for data source ([data-source-name])″

This error is denoted by message DSRA8040I: Failed to connect to the DataSource.

This error usually occurs when the class path of the DB2 JDBC driver is set correctly to `${DB2_JDBC_DRIVER_PATH}/db2java.zip` but the environment variable DB2_JDBC_DRIVER_PATH is not set.

This error can also occur if you are using DB2 Version 7.1 or 7.2 and you have not yet run *usejdbc2*. This might be the problem if your path is correct but you still receive this error.

To confirm this problem:

1. Go to the **Manage WebSphere Variables** panel.
2. Select **Environment** to verify that there is no entry for the variable DB2_JDBC_DRIVER_PATH.

To correct this problem: Add the variable DB2_JDBC_DRIVER_PATH with **value** equal to the directory path containing the `db2java.zip` file.

java.sql.SQLException: Failure in loading T2 native library db2jcct2 DSRA0010E: SQL State = null, Error Code = -99,999

The *Failure in loading* message indicates one of two things:

- Usually this happens when the machine was not rebooted after installing DB2. Reboot the machine getting the error and try it again.
- The DB2 context is not getting set up correctly for the user running WebSphere Application Server. Source the db2profile file on the machine, and ensure that the environment contains pointers to the DB2 native libraries.

Lock contention exception occurs in database when data source implementation type is XA

Note: Because a lock contention exception can be caused by many factors, consider the following explanation and recommended response as a strategy for eliminating the possible reasons for your lock contention problem.

Symptom

A lock contention exception occurs in a DB2 database that your application accesses through a data source of implementation type XA.

Problem	Your application is trying to access database records that are locked by an XA transaction that is in ended (e) state, but cannot be prepared by the transaction manager.
Description	An XA transaction to DB2 that ends, but cannot be prepared, is in ended (e) state. Because it is *not* considered to be *in doubt*, the transaction manager cannot recover this transaction. DB2 does not return it in the list of in doubt transactions. DB2 also does not roll the transaction back immediately; it waits until all connections to the database are released. During this period of inaction, the transaction continues to hold locks on the database. Due to certain policies of WebSphere Application Server workload management, your application server might not disconnect all connections from the database to allow rollback of the transaction. Therefore the ended transaction persists in locking the same database records. If your application attempts to access these locked records, a lock contention exception occurs in DB2.
Recommended response	DB2 Version 8.2 is shipped with a sample application that connects to a defined DB2 server and uses the available DB2 APIs to obtain a list of these particular ended transactions. The application offers a configuration setting that enables you to designate an amount of time after which the application rolls these transactions back. Locate the sample application in the `sqllib/samples/db2xamon.c` directory of DB2 Version 8.2 and run it.

″DSRA8050W: Unable to find the DataStoreHelper class specified″ exception occurs when trying to use a DB2 Universal Datasource in a mixed release cell.

This error usually occurs when you are using WebSphere Application Server Version 6.0 or above in conjunction with a previous version and attempt to create a DB2 Universal Datasource on the previous version.

This can happen because the DB2 Universal Datasource was not available on Version 5 and previous versions, but the Version 6 administrative console allows you to build one.

To correct this problem: create the datasource on Version 6.0 or later.

Receive ″'SYSTEM' is not a valid authorization ID″ message when trying to access DB2 on a Windows machine where WebSphere Application Server is also installed.

Symptom	For a WebSphere Application Server on Windows installation that uses DB2 as the backend, you see the following exception in the JVM log: `java.sql.SQLException: [IBM][CLI Driver] SQL0567N "SYSTEM" is not a valid authorization ID. SQLSTATE=42602 DSRA0010E: SQL State = 42602, Error Code = -567` `at COM.ibm.db2.jdbc.app.SQLExceptionGenerator.throw_SQLException (Unknown Source)` `at COM.ibm.db2.jdbc.app.SQLExceptionGenerator.check_return_code (Unknown Source)` `at COM.ibm.db2.jdbc.app.DB2Connection.connect(Unknown Source)` `at COM.ibm.db2.jdbc.app.DB2Connection.<init>(Unknown Source)` `at COM.ibm.db2.jdbc.app.DB2ReusableConnection.<init>(Unknown Source)` `at COM.ibm.db2.jdbc.DB2PooledConnection.getConnection(Unknown Source)` `at com.ibm.ws.rsadapter.spi.WSRdbDataSource.getConnection (WSRdbDataSource.java:1035)` `at com.ibm.ws.rsadapter.spi.WSManagedConnectionFactoryImpl. createManagedConnection(WSManagedConnectionFactoryImpl.java:937)` `at com.ibm.ejs.j2c.poolmanager.FreePool. createManagedConnectionWithMCWrapper(FreePool.java:1502)`
Problem	This exception occurs for configurations in which WebSphere Application Server is a client to the DB2 server. The underlying problem is an authorization conflict between WebSphere Application Server on Windows and DB2 that arises when an application attempts to connect to DB2 without providing a user ID and a password.
Description	When a DB2 client and the DB2 database run on the same machine, DB2 allows the client to connect without a user ID and password. The connection is made under the credentials of the user that owns the client process: in this case, the application server JVM. However, if WebSphere Application Server runs as a Windows service, and the ″Log on as″ option is set to ″Local System Account″, the application server JVM is categorized as a subcomponent of a special Windows user called SYSTEM. This user is not allowed to connect to DB2, resulting in the previously shown exception.
Recommended response	You have two options: • Modify the WebSphere Application Server service to use a **Log on as** option of **This account**, and provide an account with permission to connect to DB2. **Or** • Configure your application server to provide credentials on the DB2 connection by using container-managed or component-managed authentication.

XAException: XAER_NOTA on XA prepare call in DB2 Universal JDBC Driver type 4 after one phase transaction rollback

Symptom

For applications that use the DB2 Universal JDBC Driver type 4 XA available with DB2 v8.2, a connection might fail and trigger an `XAER_NOTA XAException` error. The following code block is an example of this exception:

```
J2CA0027E: An exception occurred while invoking prepare
on an XA Resource Adapter from dataSource jdbc/SDOSVT,
within transaction ID {XidImpl: formatId(57415344),
gtrid_length(36), bqual_length(54),
data(000000ff5191398200000001000000296cac5c42fe3c6838631cbaafc8b5a9253b846544
000000ff5191398200000001000000296cac5c42fe3c6838631cbaafc8b5a9253b8465440000000
10000000000000000000000000002)}:
javax.transaction.xa.XAException: XAER_NOTA
at com.ibm.db2.jcc.a.xb.a(xb.java:1682)
at com.ibm.db2.jcc.a.xb.a(xb.java:841)
```

```
at com.ibm.db2.jcc.a.xb.prepare(xb.java:812)
at com.ibm.ws.rsadapter.spi.WSRdbXaResourceImpl.prepare
(WSRdbXaResourceImpl.java:837)
...
```

Problem

If a DB2 Universal JDBC Driver type 4 XA connection is used in a single-phase transaction, such as a local transaction with autocommit set to false, and that single-phase transaction is rolled back, the next use of the connection in a two-phase transaction fails on the prepare call.

A problem in the DB2 Universal JDBC Driver type 4 XA support causes the XA prepare call to fail. This problem does not occur if the single-phase transaction is committed, and it does not occur when using the DB2 Universal JDBC Driver in type 2 mode.

Solution

Upgrade to DB2 Version 8.2 Fix Pack 1, which is equivalent to Version 8.1 Fix Pack 8. The Universal JDBC Driver XA that is available with these releases solves the previously described issue for type 4 mode.

java.rmi.MarshalException logged for application client due to incompatibility of JDBC driver file versions

Symptom

For an application that includes a J2EE application client, the following error message is displayed in the client log file of the application server:

```
java.rmi.MarshalException: CORBA MARSHAL
0x4942f89a No; nested exception is:
org.omg.CORBA.MARSHAL: Unable to read value from
underlying bridge : Mismatched serialization
UIDs : Source (Rep.
IDRMI:com.ibm.db2.jcc.c.SqlException:63EEE52211DCD763:82CE0C0DA2B0A000)
= 82CE0C0DA2B0A000 whereas Target (Rep. ID
RMI:com.ibm.db2.jcc.c.SqlException:63EEE52211DCD763:91C6171BC645E41B)
= 91C6171BC645E41B  vmcid: 0x4942f000  minor code:
2202  completed: No
```

Problem

The `db2jcc.jar` files on the application client machine and on your application server are from versions of DB2 that are not compatible with each other, or are not compatible with the version of DB2 that functions as the datastore.

Solution

Check the `db2jcc.jar` files on the application client machine, your application server, and your DB2 server. On the client machine and the application server, install files of the same version that is compatible with the DB2 server.

Database failure triggers problematic -99999 exception for applications that use DB2 Universal Driver type 4

Symptom

If you use the DB2 Universal Driver type 4 for access to DB2 or Cloudscape Network Server, and your database fails, the database server issues a generic -99999 exception in response to every JDBC getConnection request. This exception, which is exemplified in the following code excerpt, can cause unexpected behavior in your applications.

```
java.sql.SQLException: IO Exception opening socket to
 server bs8.rchland.ibm.com on port 1527.
The DB2 Server may be down.DSRA0010E: SQL State = null,
 Error Code = -99,999DSRA0010E: SQL State = null,
Error Code = -99,999
at com.ibm.db2.jcc.b.a.<init>(a.java:125)
at com.ibm.db2.jcc.b.b.a(b.java:1011)
at com.ibm.db2.jcc.c.l.<init>(l.java:197)
at com.ibm.db2.jcc.b.b.<init>(b.java:258)
at com.ibm.db2.jcc.DB2PooledConnection.
 <init>(DB2PooledConnection.java:44)
at com.ibm.db2.jcc.DB2ConnectionPoolDataSource.getPooledConnectionX
 (DB2ConnectionPoolDataSource.java:80)
at com.ibm.db2.jcc.DB2ConnectionPoolDataSource.getPooledConnection
 (DB2ConnectionPoolDataSource.java:45)
at com.ibm.ws.rsadapter.DSConfigurationHelper$1.run
 (DSConfigurationHelper.java:945)
```

Problem

When running in type 4 mode, some versions of the DB2 Universal Driver trigger a generic exception for database failure rather than a specific error code that WebSphere Application Server can map to a stale connection exception. This problem occurs with versions of the driver that are associated with DB2 8.1 Fix Pack 6 or Fix Pack 7, and DB2 8.2.

Solution

Upgrade to DB2 Version 8.2 Fix Pack 1, equivalent to Version 8.1 Fix Pack 8, which provides a valid error code in the previously described scenario. WebSphere Application Server maps this error code to a StaleConnectionException, as expected.

Cannot access DB2 on Linux when using the DB2 Universal JDBC Driver

Symptom

In the WebSphere Application Server on Linux environment, applications that use the DB2 Universal JDBC Driver to access DB2 on Linux might not connect with the database. The database server can issue the following exceptions to the application server error log:

- `java.security.AccessControlException: Access denied (java.lang.RuntimePermission accessClassInPackage.sun.io)`
- *If you run 64-bit Linux:*

  `com.ibm.db2.jcc.b.SqlException: Failure in loading T2 native library db2jcct2`

Problem

The process for configuring DB2 on Linux to work with the Universal JDBC Driver is not complete.

Solution

- Verify that the setup requirements for the Java™ SDK 1.4.2 for the Linux platform are complete.
- Configure your development environment for building Java applications on Linux with DB2 JDBC support. See the instructions at http://www.ibm.com/software/data/db2/udb/ad/v8/bldg/t0004878.htm.
- If you run DB2 on the Linux/IA64 platform and are using DB2 v8.1 Fix Pack 7A, perform the additional step that is described at http://www.ibm.com/support/docview.wss?uid=swg21189945. This step is

necessary only for Fix Pack 7A. This step is not necessary for DB2 v8.1 Fix Pack 7 or earlier versions of DB2; this step is not necessary for DB2 v8.1 Fix Pack 8 or later versions of DB2.

Test connection operation fails for DB2 legacy CLI-based Type 2 driver that is used to access DB2 running on 64-bit Linux

Symptom

If you run DB2 on 64-bit Linux using the AMD Opteron™ processor, the test connection operation fails for data sources that are configured with the DB2 legacy CLI-based Type 2 driver. You see the following error message in the WebSphere Application Server administrative console:

```
[3/8/05 16:27:19:020 CST] 0000003c DataSourceCon E DSRA8040I:
Failed to connect to the DataSource.
Encountered "": java.lang.UnsatisfiedLinkError:
COM/ibm/db2/jdbc/app/DB2Connection.SQLConnect
(Ljava/lang/String;II)I
```

Problem

Your version of 64-bit DB2, Version 8.1 Fix Pack 8, does not support the DB2 legacy CLI-based Type 2 driver on the ADM64 platform. This version of DB2 does not have the library libdb2jdbc.so, which is required by the driver.

Solution

Upgrade DB2 v8.1 to fix pack 9.

Illegal conversion occurs on any VARCHAR FOR BIT DATA column in a container-managed persistent bean

When enterprise beans with container-managed persistent (CMP) types that have any VARCHAR FOR BIT DATA columns defined on a DB2 table are deployed in the DB2 universal JDBC type 4 driver to persist the data, an SQLException of illegal conversion is thrown at run time. This exception only occurs when you use the DB2 universal JDBC type 4 driver and with the deferPrepares property being set to true. When the deferPrepares property is set to true, the DB2 universal JDBC type 4 driver uses the standard JDBC data mapping.

Currently, the generated deployed code does not follow the standard JDBC specification mapping. The failure at execution time is because of a problem in the tool that prepared the enterprise beans for execution.

To avoid receiving this exception, choose one of the following options:
- Set the deferPrepares property to false in the data source configuration.
- Do not use the DB2 universal JDBC type 4 driver if your table has any VARCHAR FOR BIT DATA or LONG VARCHAR FOR BIT DATA columns. Use the DB2 legacy CLI-based JDBC driver to persist the data. Refer to DB2 V8.1 readme for more details.

Problems accessing a SQL server data source

This article provides troubleshooting tips for accessing SQL server data sources.

What kind of problem are you having accessing your SQL Server database?

- “ERROR CODE: 20001 and SQL STATE: HY000 accessing SQLServer database” on page 793
- “Application fails with message stating ″Cannot find stored procedure...″ accessing an SQLServer database” on page 793
- “ERROR CODE: SQL5042 when running a Java application ” on page 793
- “Cannot connect to SQL Server 2000 running on Windows Server 2003 when using XA data source” on page 793

ERROR CODE: 20001 and SQL STATE: HY000 accessing SQLServer database

The problem might be that the distributed transaction coordinator service is not started. Look for an error similar to the following example when attempting to access an SQL server database:

```
ERROR CODE: 20001
SQL STATE:  HY000
java.sql.SQLException: [Microsoft][SQLServer JDBC Driver]
[SQLServer]xa_open (0) returns -3
at com.microsoft.jdbc.base.BaseExceptions.createException(Unknown Source) ...
at com.microsoft.jdbcx.sqlserver.SQLServerDataSource.getXAConnection
(Unknown Source) ...
```

To confirm this problem:

1. Go to the Windows **Control Panel** and click **Services**(or click **Control Panel > Administrative Tools > Services**)
2. Verify whether the service **Distributed Transaction Coordinator** or **DTC** is started.
3. If not, start the Distributed Transaction Coordinator service.

Application fails with message stating ″Cannot find stored procedure...″ accessing an SQLServer database

This error can occur because the stored procedures for the Java Transaction API (JTA) feature are not installed on the Microsoft SQL Server.

To resolve the problem: Repeat the installation for the stored procedures for the JTA feature, according to the ConnectJDBC installation guide.

ERROR CODE: SQL5042 when running a Java application

This error can occur when you configure your application to run in the following manner:

1. you use a type 2 (application) driver running on the gateway to the OS 390
2. your application is an XA application.

OS 390 does not use XA, but uses SPM. To resolve the problem:

1. Check your dbm cfg to see that the SPM is not started on the gateway.
2. Assign a port and set the *db2comm* variable to **TCPIP**.
3. Update the dbm cfg value *SPM_NAME* to use your machine name.
4. Start the SPM on the gateway.

Cannot connect to SQL Server 2000 running on Windows Server 2003 when using XA data source

Symptom

When using XA data source to connect to SQL Server 2000, running on Windows® 2003, you receive the following exception:

```
java.sql.SQLException: [IBM][SQLServer JDBC Driver][SQLServer]xa_open (0) returns -3
```

You might also see the following exceptions in WebSphere Application Server `SystemOut.log`:

```
[9/21/04 16:57:53:284 CST] 558bbb0a FreePool      E J2CA0046E:
Method createManagedConnctionWithMCWrapper caught an exception
during creation of the ManagedConnection for resource jdbc/lmDS,
throwing ResourceAllocationException.  Original exception:
com.ibm.ws.exception.WsException: DSRA8100E: Unable to get a
XAConnection from the DataSource.
```

```
   at
com.ibm.ws.rsadapter.exceptions.DataStoreAdapterException.<init>
(DataStoreAdapterException.java:244)
   at
  com.ibm.ws.rsadapter.exceptions.DataStoreAdapterException.<init>
(DataStoreAdapterException.java:171)
   at
com.ibm.ws.rsadapter.AdapterUtil.createDataStoreAdapterException
(AdapterUtil.java:209)
   at
com.ibm.ws.rsadapter.DSConfigurationHelper.getPooledConnection
(DSConfigurationHelper.java:911)
   at
com.ibm.ws.rsadapter.spi.WSRdbDataSource.getPooledConnection
(WSRdbDataSource.java:675)

....
Caused by: java.sql.SQLException: [IBM][SQLServer JDBC Driver]
[SQLServer]xa_open (0) returns -3
   at com.ibm.websphere.jdbc.base.BaseExceptions.createException
(Unknown Source)

---- Begin backtrace for nested exception
java.sql.SQLException: [IBM][SQLServer JDBC Driver][SQLServer]
xa_open (0) returns -3
   at com.ibm.websphere.jdbc.base.BaseExceptions.createException
(Unknown Source)
   at com.ibm.websphere.jdbc.base.BaseExceptions.getException
(Unknown Source)
```

Problem

XA transactions are disabled by default on Windows Server 2003. Microsoft® Windows Server 2003, Microsoft Distributed Transaction Coordinator (MS DTC) requires the creation of registry values for all XA DLLs that you plan to use.

Solution

Connect JDBC Drivers (DataDirect Connect JDBC, IBM® WebSphere® embedded Connect JDBC) have the XA DLL `sqljdbc.dll` that is normally installed on `SQLServer_Install_Root\MSSQL\Binn`. For example: `c:\Program Files\Microsoft SQL Server\MSSQL\Binn`) The registry values required for XA transactions are not created automatically. You must create the values manually as follows:

1. Turn on support for XA transactions:
   a. Open Component Services.
   b. Expand the tree view to locate the computer where you want to turn on support for XA transactions; for example, My Computer.
   c. Right-click the computer name, then click **Properties**.
   d. Click the MSDTC tab, then click **Security Configuration**.
   e. Under Security Settings, click the check box for XA Transactions to turn on this support.
   f. Click **OK**, then click **OK** again.
2. Create a registry named-value:
   a. Use Registry Editor and navigate to registry key:
      `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\XADLL`
   b. Create a new registry named-value:
      - **Name** is the file name of the XA DLL (in the format dllname.dll)
      - **Type** is String (REG_SZ)
      - **Value** is the full path name (including the file name) of the DLL file

Name	Type	Value
sqljdbc.dll	String (REG_SZ)	c:\Program Files\Microsoft SQL Server\MSSQL\Binn\sqljdbc.dll

Note: You must create an entry for each XA DLL file that you plan to use. Also, if you are configuring MS DTC on a cluster, you must create these registry entries on each node in the cluster.

For more details, see the following Microsoft document:

INFO: Registry Entries Are Required for XA Transaction Support

Problems accessing a Cloudscape database

This article provides troubleshooting tips for accessing Cloudscape databases.

What kind of problem are you having accessing your Cloudscape database?

- "Unexpected IOException wrapped in SQLException, accessing Cloudscape database"
- "The ″select for update″ operation causes table lock and deadlock when accessing Cloudscape"
- "ERROR XSDB6: Another instance of Cloudscape may have already booted the database ″database″"
- "Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Cloudscape databases″" on page 796
- "Running an application causes a runtime exception which produces an unreadable message." on page 796

Tip: Cloudscape errorCodes (2000, 3000, 4000) indicate levels of severity, not specific error conditions. In diagnosing Cloudscape problems, pay attention to the given sqlState value.

Unexpected IOException wrapped in SQLException, accessing Cloudscape database

This problem can occur because Cloudscape databases use a large number of files. Some operating systems, such as the Solaris Operating Environment, limit the number of files an application can open at one time. If the default is a low number, such as 64, you can get this exception.

If your operating system lets you configure the number of file descriptors, you can correct the problem by setting the number to a high value, such as 1024.

The ″select for update″ operation causes table lock and deadlock when accessing Cloudscape

If a select for update operation on one row locks the entire table, which creates a deadlock condition, there might be undefined indexes on that table. The lack of an index on the columns you use in the `where` clause can cause Cloudscape to create a table lock rather than a row level lock.

To resolve this problem, create an index on the affected table.

ERROR XSDB6: Another instance of Cloudscape may have already booted the database ″database″

This problem occurs because Cloudscape embedded framework only allows one Java virtual machine (JVM) to access the database instance at a time.

To resolve this problem:

1. Verify that you do not have other JDBC client programs, such as **ij** or **cview** running on that database instance, when WebSphere Application Server is running.

2. Verify that you do not use the same instance of the database for more than one data source or use the networkServer framework, which doesn't have this limitation.
3. If there are no connections to Cloudscape, delete the db.lck lock file. This file can be found in the directory where the Cloudscape database is mounted, under the schema directory. For example, if the database is mounted at /myCloudscapeDB, issue the command: rm /myCloudscapeDB/schemaName/db.lck

Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Cloudscape databases″

Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Cloudscape databases″

At the client runtime, an error similar to the following occurs:

```
The version of the IBM Universal JDBC driver in use is not
licensed for connectivity to Cloudscape databases.  To connect
to this DB2 server, please obtain a licensed copy of the IBM DB2
Universal Driver for JDBC and SQLJ.  An appropriate license file
db2jcc_license_*.jar for this target platform must be installed to
the application classpath.  Connectivity to Cloudscape databases is
enabled by any of the following license files:
{ db2jcc_license_c.jar, b2jcc_license_cu.jar, db2jcc_license_cisuz.jar }
```

The problem occurs because an incorrect JDBC driver jar file name is specified in the class path for JDBC provider. For example, the jar file name may have an extra '_', as follows:

```
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license__cu.jar
```

To resolve the problem:

1. Correct the UNIVERSAL_JDBC_DRIVER_PATH jar file name in the JACL script
2. Restart the cluster.
3. Rerun the client.

Running an application causes a runtime exception which produces an unreadable message.

At client runtime, you may receive a message similar to the following: `Caused by: com.ibm.db2.jcc.a.SqlException: DB2 SQL error: SQLCODE: -1, SQLSTATE: 42X05, SQLERRMC: ANNUITYHOLDER20^T42X05`

The problem occurs because the property *retrieveMessagesfromServerOnGetMessage*, which is required by WebSphere Application Server, has not been set.

To resolve the problem, on the admininstrative console

1. Click **Resources -> JDBC Providers**
2. Click on a Cloudscape provider
3. Scroll down and click on **Data Sources**
4. Select your data source (or add a new one)
5. Scroll down and select **Custom Properties**
6. If the property *retrieveMessagesFromServerOnGetMessage* already exists, set its value to true. If the property does not exist, select **New**and add the property *retrieveMessagesFromServerOnGetMessage* with a value **true**
7. Rerun the client

The SystemOut.log will now generate readable messages so that you can resolve the underlying problem.

Problems accessing a Sybase data source

This article provides troubleshooting tips for accessing Sybase data sources.

What kind of problem are you having accessing your Sybase database?

- ″Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode″ error.
- ″JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.″
- A container managed persistence (CMP) enterprise bean is causing exceptions.
- “Sybase JDBC data source fails with ″Incorrect URL format″ exception in IPv6 environment” on page 798
- “Executing the DatabaseMetaData.getBestRowIdentifier() method in an XA transaction causes errors” on page 798
- “Sybase requirements for using the escapes and DatabaseMetaData methods” on page 799
- “Database deadlocks and XA_PROTO errors occur when using Sybase” on page 799
- “Executing a stored procedure containing a SELECT INTO command causes exception” on page 800
- “Error is incorrectly reported about IMAGE to VARBINARY conversion” on page 800
- “JDBC 1.0 standard methods are not implemented and generate a SQL exception when used” on page 800
- “Sybase transaction manager fails after trying to alleviate a deadlock error” on page 800
- “Starting an XA transaction when the autoCommit value of the connection is false causes error” on page 800
- “Sybase does not throw an exception when an incorrect database name is specified” on page 801

″Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode″ error

This error occurs when either:

- The JDBC attempts to put the connection in **autocommit(true)** mode.
- A stored procedure is not created in a compatible mode.

To fix the **autocommit(true)** mode problem, let the application change the connection to chained mode using the **Connection.setAutoCommit(false)** mode, or use a **set chained on** language command.

To resolve the stored procedure problem, use the `sp_procxmode` *`procedure_name`* `"anymode"` command.

″JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.″

This error occurs when XA-style transactions are attempted on a server that does not have Distributed Transaction Management (DTM) installed.

To resolve this problem, use the instructions in the Sybase Manual titled: *Using Adaptive Server Distributed Transaction Management Features* to enable Distributed Transaction Management (DTM). The main steps in this procedure are:

1. Install the DTM option.
2. Check the `license.dat` file to verify that the DTM option is installed.
3. Restart the license manager.
4. Enable DTM in ISQL.
5. Restart the ASE service.

A container managed persistence (CMP) enterprise bean is causing exceptions

This error is caused by improper use of reserved words. Reserved words cannot be used as column names.

To correct this problem: Rename the variable to remove the reserved word. You can find a list of reserved words in the *Sybase Adaptive Server Enterprise Reference Manual; Volume 1: Building Blocks*, Chapter 4. This manual is available online at: http://manuals.sybase.com/onlinebooks/group-as/asg1250e/refman.

Sybase JDBC data source fails with ″Incorrect URL format″ exception in IPv6 environment

Problem

If you configure a Sybase JDBC data source through the administrative console and attempt to use it in an IPv6 environment, you can experience the following exception:

```
java.sql.SQLException: JZ0NE: Incorrect URL format
```

Cause

The Sybase JDBC drivers that are listed as selections for JDBC provider type in the administrative console do not support IPv6. The administrative console does not contain a pre-formatted template for the Sybase jConnect JDBC driver v6.0 EBF12884, which is the version required to connect to the database in an IPv6 environment.

However, you can still use the administrative console to define the Sybase jConnect JDBC driver v6.0 EBF12884.

Solution

Complete the following steps to define the Sybase jConnect JDBC driver v6.0 EBF12884:

1. In the administrative console, go to **Resources** > **JDBC Providers** and click **New**.
2. Select **User-defined** for the database type. This selection triggers the console to set your provider type to User-defined JDBC provider and implementation type to User-defined.
3. Click **Next** to go to the JDBC provider general configuration page.
4. In the Class path field, replace the default JAR file name with `jconn3.jar`.
5. For Implementation class name:
   - For a data source implementation that supports only one phase transactions, input
     `com.sybase.jdbc3.jdbc.SybConnectionPoolDataSource`
   - For an implementation that supports two phase transactions, input
     `com.sybase.jdbc3.jdbc.SybXADataSource`
6. Click **Apply**. Then click the Data sources link.
7. Click **New** to define the general properties of your data source.
8. Click **Apply** after defining the general data source properties. Then click the Custom properties link.
9. Define the following properties as custom properties. To set these properties, click **New**. Input the property name and a valid value for each one.
   - databaseName
   - serverName
   - portNumber

Executing the DatabaseMetaData.getBestRowIdentifier() method in an XA transaction causes errors

Executing the DatabaseMetaData.getBestRowIdentifier() method while in an XA transaction causes the following errors:

```
SQL Exception: The 'CREATE TABLE' command is not allowed within a
multi-statement transaction in the 'tempdb' database.  Calling
DatabaseMetaData.getBestRowIdentifier()
```

Currently, this method fails when using Sybase. This problem occurs with other methods as well, including:
- getBestRowIdentifier();
- getVersionColumns();
- getTablePrivileges();
- getProcedureColumns();
- getPrimaryKeys();
- getIndexInfo();
- getImportedKeys();
- getExportedKeys();
- getCrossReference();
- getColumns();
- getColumnPrivileges();

Case 10880427 has been opened with Sybase to resolve this problem.

Sybase requirements for using the escapes and DatabaseMetaData methods

To use the escapes and DatabaseMetaData methods, you must install stored procedures on the Adaptive Server Enterprise or Adaptive Server Anywhere database where you want to use these methods. These stored procedures are also required by some of the connection methods.

To check for the presence of LOCATE ():
1. Open a Sybase **isql** command prompt.
2. Type the command **use master**.
3. Type the command **go**.
4. Type the SQL command and select * from jdbc_function_escapes.
5. Type the command **go**.

The following appears:

```
escape_name      map_string
 --------------------------------
 abs                    abs(%1)
 acos                  acos(%1)
 asin                   asin(%1)
 atan                   atan(%1)
 atan2                 atn2(%1, %2)
 ceiling                ceiling(%1)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::

locate charindex ((convert (varchar, %1)), (convert (varchar, %2)))
```

If the function does not exist, upgrade jConnect to at least Version 5.2 EBF 10635 and run the following command:

```
java IsqlApp -U sa -P -S jdbc:sybase:Tds:hostname:4100 -I %JDBC_HOME%\sp\sql_server12.sql -c go
```

Database deadlocks and XA_PROTO errors occur when using Sybase

When using Sybase with the IBM WebSphere Application Server, do one of the following to prevent database deadlocks and errors:
- Change the transaction isolation level on the connection to TRANSACTION_READ_COMMITTED. Set the isolation level on the connection for unshareable connections or, for shareable connections, define the isolation levels in the resource reference for your data source using an assembly tool.
- Modify Sybase by doing one of the following:
  - If you want to use the existing tables, modify the table locking scheme using the **alter table *table name* lock datarows** command to get a row lock level granularity.
  - If you want to set the system-wide locking scheme to data rows, all subsequently created tables inherit that value and have a locking scheme of data rows.

Note: You must drop your original databases and tables.

Executing a stored procedure containing a SELECT INTO command causes exception

An attempt to execute a stored procedure containing a **SELECT INTO** command results in the following exception:

```
SVR-ERROR: SQL Exception SELECT INTO command not allowed within multi-statement transaction
```

Case 10868947 has been opened with Sybase to resolve this problem.

Error is incorrectly reported about IMAGE to VARBINARY conversion

The following error is incorrectly reported:

```
com.sybase.jdbc2.jdbc.SybSQLException: Implicit conversion from data type 'IMAGE' to 'VARBINARY' is not allowed.
 Use the CONVERT function to run this query.
```

The error is about a VARBINARY column only and causes confusion if you also have an IMAGE column.

Do one of the following to work around this problem:
- Use a PreparedStatement.setBytes() method instead of a PreparedStatement.setBinaryStream() method
- Use a LONG VARBINARY for the column type if you want to continue using the setBinaryStream() method. You might want to make this change because the size limit for VARBINARY is 255 bytes.

For example:

```
// ***************CORRECTION****************************
         // setBinaryStream fails for column type of VARBINARY , use setBytes() instead
         //stmt4.setBinaryStream(8,new java.io.ByteArrayInputStream(tempbyteArray),tempbyteArray.length);
         stmt4.setBytes(8,tempbyteArray);
```

JDBC 1.0 standard methods are not implemented and generate a SQL exception when used

The following JDBC 1.0 standard methods are not implemented and generate a SQL exception when used:
- ResultSetMetaData.getSchemaName()
- ResultSetMetaData.getTableName() (implemented only for text and image datatypes)
- ResultSetMetaData.getCatalogName()

Sybase transaction manager fails after trying to alleviate a deadlock error

If an application encounters a deadlock, Sybase detects the deadlock and throws an exception. Because of this detection, the transaction manager calls an `xa_end` with a `TMFAIL` in it.

The call succeeds, but causes another Sybase exception, `XAERR_PROTO`. This exception only appears in the error log and does not cause any functional problems. All applications should continue to run, therefore no workaround is necessary.

Case 10869169 has been opened with Sybase to resolve this problem.

Starting an XA transaction when the autoCommit value of the connection is *false* causes error

The exception thrown is `javax.transaction.xa.XAException` with stack trace similar to the following:

```
at com.sybase.jdbc2.jdbc.SybXAResource.sendRPC(SybXAResource.java:711)
 at com.sybase.jdbc2.jdbc.SybXAResource.sendRPC(SybXAResource.java:602)
 at com.sybase.jdbc2.jdbc.SybXAResource.start(SybXAResource.java:312)
```

This problem affects you when you do both local and global transactions. If, in a local transaction, the autoCommit default value is set to *false*, and a global or XA transaction starts (either a user transaction started by you, or a container transaction started by a container), the exception occurs.

This problem is a Sybase bug as the start() method can fail unexpectedly, regardless of the value of autoCommit. Currently, there is no workaround for this problem, therefore it is not recommended that you mix local and global transactions. Case 10880792 has been opened to resolve this problem.

Sybase does not throw an exception when an incorrect database name is specified

Verify that your database name is correctly entered on the data source properties.

Most databases (DB2, Oracle, Informix , MS SQL Server and Cloudscape) throw an exception when the database specified does not exist. But Sybase does not throw an exception when an incorrect database name is specified. Sybase generates an SQL warning and then connects to the default database. If you misspell the requested database name, Sybase connects you to the master or the default database where the table you requested is not found.

If none of these steps fixes your problem, 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 do not find your problem listed there, contact IBM Support.

JDBC trace configuration

If your application displays JDBC-related exception messages, activate the JDBC trace service. The resulting log text can help you identify the problem.

Trace strings for JDBC data sources

Turn on tracing for most database JDBC implementations through the administrative console; see the article Enabling trace at server startup for instructions. This method activates JDBC trace for all applications that run in the server you specify. Identify your database type by selecting the trace group WAS.database and typing one of the following trace strings in the console:

- **com.ibm.ws.database.logwriter** Trace string for databases that use the GenericDataStoreHelper. You can also use this trace string for unsupported databases.
- **com.ibm.ws.db2.logwriter** Trace string for DB2 databases.
- **com.ibm.ws.oracle.logwriter** Trace string for Oracle databases.
- **com.ibm.ws.derby.logwriter** Trace string for Derby databases.
- **com.ibm.ws.informix.logwriter** Trace string for Informix databases.
- **com.ibm.ws.sqlserver.logwriter** Trace string for Microsoft SQL Server databases.
- **com.ibm.ws.sybase.logwriter** Trace string for Sybase databases.

A few JDBC drivers require that you set trace differently, at the data source level. These drivers include:

- DB2 legacy CLI-based JDBC driver
- WebSphere embedded ConnectJDBC driver for MS SQL Server
- DataDirect ConnectJDBC driver for MS SQL Server
- DataDirect SequeLink JDBC driver for MS SQL Server, which is deprecated in WebSphere Application Server V6.0
- Microsoft JDBC driver for MS SQL Server 2000, which is deprecated in WebSphere Application Server V6.0

Configuring trace for these drivers through the WAS.database group results in corrupt trace information. WebSphere Application Server sets trace for the group at the server level, causing the trace service to begin only after your application establishes an initial connection. Because that first connection does not carry trace information, re-use of it is never tracked. Consequently the application cannot accurately match trace information to connection use.

Set trace for the previously mentioned JDBC drivers through data source custom properties. For example, use custom property `spyAttributes` to enable the JDBC trace for SequeLink or Connect JDBC drivers. Consult your driver documentation for details on the custom property that enables trace for your JDBC implementation.

Additional resources

If the JDBC tracing service cannot help you isolate and fix your problem, consult the IBM Support website for WebSphere Application Server at http://www.ibm.com/support/search.wss?rs=180&tc=SSEQTP&tc1=SSCMP9Y. Use the site search function to find current information on known problems and their resolutions. Locating the right troubleshooting tip can save time that, otherwise, you might spend on opening and tracking a PMR.

Learn about other IBM Support options in the articles Diagnosing and fixing problems: Resources for learning and Troubleshooting help from IBM.

Deploying data access applications

Frequently, deploying data access applications involves more than installing your WAR or EAR file onto a server. Deployment can include tasks for configuring your application to use the data access resources of the server and overall run-time environment.

You can only deploy application code that is assembled into the appropriate modules. The topic Assembling data access applications provides guidelines for this process.

Perform the following steps if your application requires access to a relational database (RDB). If your application requires access to a different type of enterprise information system (EIS), such as an object-oriented database or the Customer Information Control System (CICS), consult the topics “J2EE Connector Architecture resource adapters” on page 558 and Accessing data using J2EE Connector Architecture connectors.

1. If your RDB configuration does not already exist:
   a. Create a database to hold the data.
   b. Create tables required by your application.
      If your application uses CMP entity beans to access the data
      You can create the tables using the data definition language (DDL) generated from the enterprise bean configuration. For more information, see Recreating database tables from the exported table data definition language.
      If your application uses BMP entity beans, or *does not* **use entity beans**
      You must use your database server interfaces to create the tables.

      You can also use the EJB to RDB Mapping wizard of an assembly tool to create your database tables for either type of entity bean. Select the top-down mapping option in the wizard. Keep in mind, however, that this option does not give you direct control in naming the RDB elements or choosing column types. Additionally, because the top-down process is automatic, it might not provide mappings to reflect the precise relationships that you intend.

      If you use the Application Server Toolkit (AST), consult the that product information center about the mapping wizard. To learn about all of your assembly tool options, consult the Assembly tools article in this information center.
   c. Check Minimum required properties for vendor-specific data sources to see any database vendor requirements for connecting to an application server.
2. If necessary, map your entity beans to the database tables through the meet-in-the-middle mapping option of an assembly tool. This step is necessary only if you did not create your database schema through the top-down mapping option, did not generate your mapping relationships through bottom-up mapping, or did not generate mappings during the application assembly process. For information on the top-down mapping option refer to the **Application Server Toolkit** information center.

3. Install your application onto the application server. Consult “Installing application files” on page 28. When you install the application, you can alter data access settings that were made during application assembly, or set them for the first time if they were omitted from the assembly process. These settings include resource bindings and resource authentication aliases, which are addressed in the following substeps:
   a. Bind application resource references to the data sources, or other resource objects, that provide database connectivity. For details on the concept of binding, see the Data source lookups for enterprise beans and Web modules topic.

      Tip: After deployment, you can use the WebSphere Application Server administrative console to alter resource bindings. Click **Applications** > **Enterprise Applications** > *application_name*, and select the link to the appropriate mapping page. For example, if you want to alter the binding of an EJB module resource, you might click **1.x CMP bean data sources** or **2.x CMP bean data sources**. For a Web module resource, click **Resource references**.
   b. Define authentication alias data for resources that must be authenticated with the backend through *container-managed* authorization. In this security configuration, WebSphere Application Server performs EIS signon for data source or connection factory connections. Consult the “J2EE connector security” on page 682 topic for detailed reference on resource authentication.
4. Start the deployed application files using the administrative console , the wsadmin startApplication command, or your own Java program.
5. Save the changes to your administrative configuration.
6. Test the application. For example, point a Web browser at the URL for a deployed application and examine the performance of the application.

If the application does not perform as desired, update the application, then save and test it again.

Available resources

Use this page to select configured resources that you want to bind to the resource references of the enterprise beans or Web modules in your application.

To view this administrative console page:

1. Click **Applications** > **Enterprise Applications** > *Application_name*.
2. Click the link for any of these resource configuration pages:
   - **Resource references**
   - **Map data sources for all 2.x CMP beans**
   - **Provide default data source mapping for modules containing 2.x entity beans**
3. Locate the table row of the EJB or Web module that you want to map to a different resource.
4. Within the row, locate the JNDI name of the resource that is currently bound to the EJB or Web module.
5. Click **Browse**.
   You now see **Available resources**.

Each table row corresponds to a resource that you can bind to your enterprise bean or Web module.

Select

Select the resource that you want to bind to the resource reference of your module.

JNDI name

The Java Naming and Directory Interface (JNDI) name of the resource that you want to bind to the resource reference of your module.

Data type String

Scope
The scope of the resource. Note that this administrative console page displays only resources that are configured for a scope at which your application operates.

Description
The text description of the resource.

1.x CMP bean data sources
Use this page to designate how the container-managed persistence (CMP) 1.x beans of an application map to data sources that are available to the application.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application_name* > **1.x CMP bean data sources**.

Guidelines for using this administrative console page:

- The table depicts the 1.x CMP bean contents of your application.
- 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 or installation. For every data source that is displayed, you see the corresponding security configuration.
- To set your 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 CMP beans.
  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 CMP beans.
  3. Click **Apply**. The console displays the 1.x CMP 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 data source security settings:
  1. Select one or more rows in the table.
  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 J2EE Connector (J2C) authentication data list, you must input them into the list after saving your settings on this page. Read the Managing J2EE Connector Architecture authentication data entries information center topic for instruction.
  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 settings.

Table column heading descriptions:

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.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the data source that is configured for the enterprise bean.

Data type String

User name
The user name and password that comprise the authentication alias for securing the data source.

1.x entity bean data sources
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** > **Enterprise Applications** > *application_name* > **1.x entity bean data sources**.

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 J2EE Connector (J2C) authentication data list, you must input them into the list after saving your settings on this page. Read the Managing J2EE Connector Architecture authentication data entries information center topic for instruction.
  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.

Table column heading descriptions:

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.

Data type String

User name
The user name and password that comprise the authentication alias for securing the data source.

2.x CMP bean data sources
Use this page to designate how the container-managed persistence (CMP) 2.x beans of an application map to data sources that are available to the application.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application_name* > **2.x CMP bean data sources**.

Guidelines for using this administrative console page:

- The page displays a table that depicts the 2.x CMP bean contents of your application.
- 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.
- To set your 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 CMP beans.
  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 CMP beans.
  3. Click **Apply**. The console displays the 2.x CMP 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 define data source security:
  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. 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. Click **Apply**.
  3. To modify the authorization method of a data source with container-managed authorization, you have three options: None, Default, or Custom login configuration. The reconfiguring process differs slightly for each option:
     – If you select **None**:
       a. Determine which data source configurations to designate with no authentication method.
       b. Select the appropriate table rows.
       c. Select **None** from the list of authentication method options that precede the table.
       d. Click **Apply**.

- If you select **Default**:
  a. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. You must apply this option to each data source individually if you want to designate different authentication data aliases. See the ″J2EE Connector security″ information center topic for more information on the default mapping configuration.
  b. Select the appropriate table rows.
  c. Select **Use default method** from the list of authentication method options that precede the table.
  d. Select an authentication data entry or alias from the list.
  e. Click **Apply**.
- If you select **Custom login configuration**:
  a. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the ″J2EE Connector security″ information center topic for more information on custom JAAS login configurations.
  b. Select the appropriate table row.
  c. Select **Use custom login configuration** from the list of authentication method options that precede the table.
  d. Select an application login configuration from the list.
  e. Click **Apply**.
  f. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

- Repeat all of the previous steps as necessary.
- Click **OK** to save your work. 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.

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.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the data source that is configured for the enterprise bean.

Data type String

Resource authorization
The authorization type and the authentication method for securing the data source.

2.x entity bean data sources

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 page, click **Applications** > **Enterprise Applications** > *application_name* > **2.x entity bean data sources**.

Guidelines for using this administrative console page:

- The page 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.
- 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 2.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 define data source security:
  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. 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. Click **Apply**.
  3. To modify the authorization method of a data source with container-managed authorization, you have three options: None, Default, or Custom login configuration. The reconfiguring process differs slightly for each option:
     - If you select **None**:
       a. Determine which data source configurations to designate with no authentication method.
       b. Select the appropriate table rows.
       c. Select **None** from the list of authentication method options that precede the table.
       d. Click **Apply**.
     - If you select **Default**:
       a. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. You must apply this option to each data source individually if you want to designate different authentication data aliases. See the ″J2EE Connector security″ information center topic for more information on the default mapping configuration.
       b. Select the appropriate table rows.
       c. Select **Use default method** from the list of authentication method options that precede the table.
       d. Select an authentication data entry or alias from the list.
       e. Click **Apply**.
     - If you select **Custom login configuration**:

a. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the ″J2EE Connector security″ information center topic for more information on custom JAAS login configurations.
b. Select the appropriate table row.
c. Select **Use custom login configuration** from the list of authentication method options that precede the table.
d. Select an application login configuration from the list.
e. Click **Apply**.
f. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

- Repeat all of the previous steps as necessary.
- Click **OK** to save your work. You now return to the general configuration page for your enterprise application.

Table column heading descriptions:

Select
Select the check boxes of the rows you want to edit.

EJB Module
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
The Java Naming and Directory Interface (JNDI) name of the default data source for the EJB module.

Data type String

Resource authorization
The authorization type and the authentication method for securing the data source.

Chapter 14. Messaging resources

Using asynchronous messaging

These topics describe how enterprise applications can use asynchronous messaging as a method of communication based on the Java Message Service (JMS). With the support provided by WebSphere Application Server, applications can make use of JMS resources and message-driven beans.

WebSphere Application Server support for JMS is provided by one or more *JMS providers*, and associated services and resources, that you configure for use by enterprise applications. You can deploy EJB 2.1 applications that use the JMS 1.1 interfaces and EJB 2.0 applications that use the JMS 1.0.2 interfaces.

You can use the WebSphere administrative console to administer the WebSphere Application Server support for asynchronous messaging. For example, you can configure messaging providers and their resources, and can control the activity of messaging services.

For more information about implementing WebSphere enterprise applications that use asynchronous messaging, see the following topics:
- Learning about messaging with WebSphere
- Installing a messaging provider
- Using the default messaging provider
- “Maintaining Version 5 default messaging resources” on page 850
- “Using the JMS resources provided by WebSphere MQ” on page 883
- “Using JMS resources of a generic provider” on page 966
- “Administering support for message-driven beans” on page 975
- Programming to use asynchronous messaging
- Troubleshooting WebSphere messaging

Learning about messaging with WebSphere Application Server

Use this topic to learn about the use of asynchronous messaging for enterprise applications with WebSphere Application Server.

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) and Java Connector Architecture (JCA) programming interfaces. These interfaces provide a common way for Java programs (clients and J2EE applications) to create, send, receive, and read asynchronous requests, as messages. For additional information about messaging resources, see: Messaging resources.
- JMS providers
- Styles of messaging in applications
- JMS interfaces - explicit polling for messages
- Message-driven beans - automatic message retrieval
- “Configuring JMS resources for the WebSphere MQ messaging provider” on page 956
- Components of message-driven bean support
- Security considerations for asynchronous messaging

JMS providers

This topic provides an overview of the support for JMS providers by WebSphere Application Server.

Overview

WebSphere Application Server supports asynchronous messaging through the use of a JMS provider and its related messaging system. JMS providers must conform to the JMS specification version 1.1. To use

message-driven beans the JMS provider must support the optional Application Server Facility (ASF) function defined within that specification, or support an inbound resource adapter as defined in the JCA specification version 1.5.

WebSphere Application Server supports JMS messaging using the following:

- Service integration default messaging provider
- WebSphere MQ JMS provider
- Version 5 default provider
- Generic JMS provider

WebSphere applications can use messaging resources provided by any of these JMS providers. However the choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you may already have a messaging infrastructure based on WebSphere MQ. In this case you may either connect directly using the included support for WebSphere MQ as a JMS provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

Service integration default provider

The service integration technologies of WebSphere Application Server can act as a messaging system when you have configured a service integration bus that is accessed through the default messaging provider. This support is installed as part of WebSphere Application Server, administered through the administrative console, and is fully integrated with the WebSphere Application Server runtime.

WebSphere MQ JMS provider

WebSphere Application Server also includes support for the WebSphere MQ JMS provider. This is provided for use with supported versions of WebSphere MQ.

Version 5 default provider

For backwards compatibility with earlier releases, WebSphere Application Server also includes support for the V5 default messaging provider which enables you to configure resources for use with the WebSphere Application Server version 5 Embedded Messaging system. The V5 default messaging provider can also be used with a service integration bus.

The V5 default messaging provider is the version 5 embedded WebSphere MQ provider. It is designed for use with applications that still use version 5 resources to communicate with version 5 nodes in mixed version cells that use embedded messaging.

Generic JMS provider

WebSphere Application Server also includes support for the generic JMS provider. This is provided for use with any third party messaging system. If you want to use message-driven beans, the messaging system must support ASF.

For additional information about connecting to WebSphere MQ, see Ways of interoperating with WebSphere MQ

For more information about connecting to WebSphere MQ, see Learning about WebSphere MQ server

Styles of messaging in applications

This topic describes the ways that applications can use point-to-point and publish/subscribe messaging.

Applications can use the following styles of asynchronous messaging:

Point-to-Point

Point-to-point applications use *queues* to pass messages between each other. The applications are called point-to-point, because a client sends a message to a specific queue and the message is picked up and processed by a server listening to that queue. It is common for a client to have all its messages delivered to one queue. Like any generic mailbox, a queue can contain a mixture of messages of different types.

Publish/subscribe

Publish/subscribe systems provide named collection points for messages, called *topics*. To send messages, applications publish messages to topics. To receive messages, applications subscribe to topics; when a message is published to a topic, it is automatically sent to all the applications that are subscribers of that topic. By using a topic as an intermediary, message publishers are kept independent of subscribers.

Both styles of messaging can be used in the same application.

Applications can use asynchronous messaging in the following ways:

One-way

An application sends a message, and does not want a response. This pattern of use is often referred to as a *datagram*.

Request / response

An application sends a request to another application and expects to receive a response in return.

One-way and forward

An application sends a request to another application, which sends a message to yet another application.

These messaging techniques can be combined to produce a variety of asynchronous messaging scenarios.

For more information about how such messaging scenarios are used by WebSphere enterprise applications, see the following topics:

- An overview of asynchronous messaging with JMS
- An overview of asynchronous messaging with message-driven beans

For more information about these messaging techniques and the Java Message Service (JMS), see Sun's Java Message Service (JMS) specification documentation (http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/).

For more information about message-driven bean and inbound messaging support, see Sun's Enterprise JavaBeans specification (http://java.sun.com/products/ejb/docs.html).

For information about JCA inbound messaging processing, see Sun's J2EE Connector Architecture specification (http://java.sun.com/j2ee/connector/download.html).

JMS interfaces - explicit polling for messages

This topic provides an overview of applications that use JMS interfaces to explicitly poll for messages on a destination then retrieve messages for processing by business logic beans (enterprise beans).

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interfaces. JMS provides a common way for Java programs (clients and J2EE applications) to create, send, receive, and read asynchronous requests, as JMS messages.

The base support for asynchronous messaging using JMS, shown in the following figure, provides the common set of JMS interfaces and associated semantics that define how a JMS client can access the facilities of a JMS provider. This enables WebSphere J2EE applications, as JMS clients, to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics).

Applications can use both point-to-point and publish/subscribe messaging (referred to as “messaging domains” in the JMS specification), while supporting the different semantics of each domain.

WebSphere Application Server supports applications that use JMS 1.1 domain-independent interfaces (referred to as the “common interfaces” in the JMS specification). With JMS 1.1, the preferred approach for implementing applications is to use the common interfaces. The JMS 1.1 common interfaces provide a simpler programming model than domain-specific interfaces. Also, applications can create both queues and topics in the same session and coordinate their use in the same transaction.

The common interfaces are also parents of domain-specific interfaces. These domain-specific interfaces (provided for JMS 1.0.2 in WebSphere Application Server version 5) are supported only to provide inter-operation and backward compatibility with applications that have already been implemented to use those interfaces.

A WebSphere application can use the JMS interfaces to explicitly poll a JMS destination to retrieve an incoming message, then pass the message to a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to send the message on to another JMS destination.

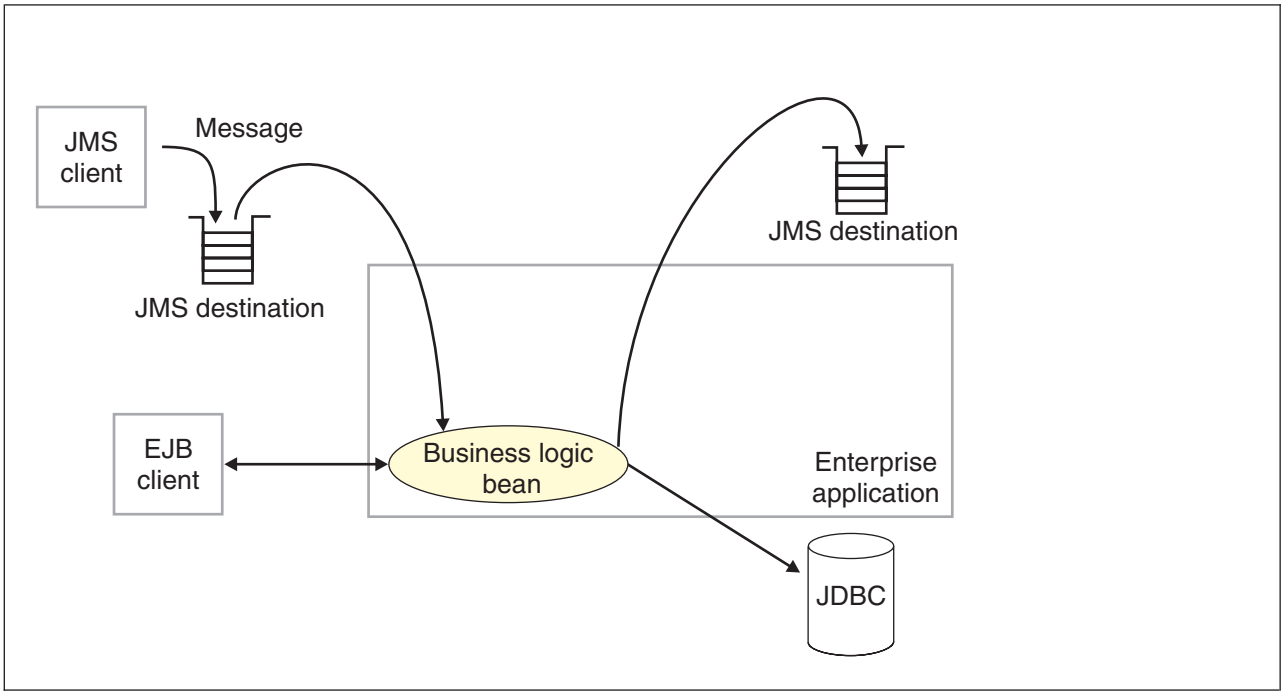


Figure 3. Asynchronous messaging using JMS. This figure shows an enterprise application polling a JMS destination to retrieve an incoming message, which it processes with a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to send the message on to another JMS destination. For more information, see the text that accompanies this figure.

WebSphere applications can use standard JMS calls to process messages, including any responses or outbound messaging. Responses can be handled by an enterprise bean acting as a sender bean, or handled in the enterprise bean that receives the incoming messages. Optionally, this process can use two-phase commit within the scope of a transaction. This level of functionality for asynchronous messaging

is called *bean-managed messaging*, and gives an enterprise bean complete control over the messaging infrastructure; for example, for connection and session pool management. The application server has no role in bean-managed messaging.

WebSphere applications can also use message-driven beans, as described in related topics.

For more details about JMS, see Sun's Java Message Service (JMS) specification documentation.

Message-driven beans - automatic message retrieval

WebSphere Application Server supports the use of message-driven beans as asynchronous message consumers.

Messaging with message-driven beans is shown in the figure Messaging with message-driven beans.

A client sends messages to the destination (or endpoint) for which the message-driven bean is deployed as the message listener. When a message arrives at the destination, the EJB container invokes the message-driven bean automatically without an application having to explicitly poll the destination. The message-driven bean implements some business logic to process incoming messages on the destination.

Message-driven beans can be configured as listeners on a Java Connector Architecture (JCA) 1.5 resource adapter or against a listener port (as in WebSphere Application Server version 5). With a JCA 1.5 resource adapter, message-driven beans can handle generic message types, not just JMS messages. This makes message-driven beans suitable for handling generic requests inbound to WebSphere Application Server from enterprise information systems through the resource adapter. In the JCA 1.5 specification, such message-driven beans are commonly called *message endpoints* or simply *endpoints*.

All message-driven beans must implement the MessageDrivenBean interface. For JMS messaging, a message-driven bean must also implement the message listener interface, javax.jms.MessageListener.

A message driven bean can be registered with the EJB timer service for time-based event notifications if it implements the javax.ejb.TimedObject interface in addition to the message listener interface.

You are recommended to develop a message-driven bean to delegate the business processing of incoming messages to another enterprise bean, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client.

Messages arriving at a destination being processed by a message-driven bean have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see EJB component security.

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, such as the default messaging provider that is part of WebSphere Application Server version 6. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA 1.5-compliant resources, to use a J2C activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server version 5).

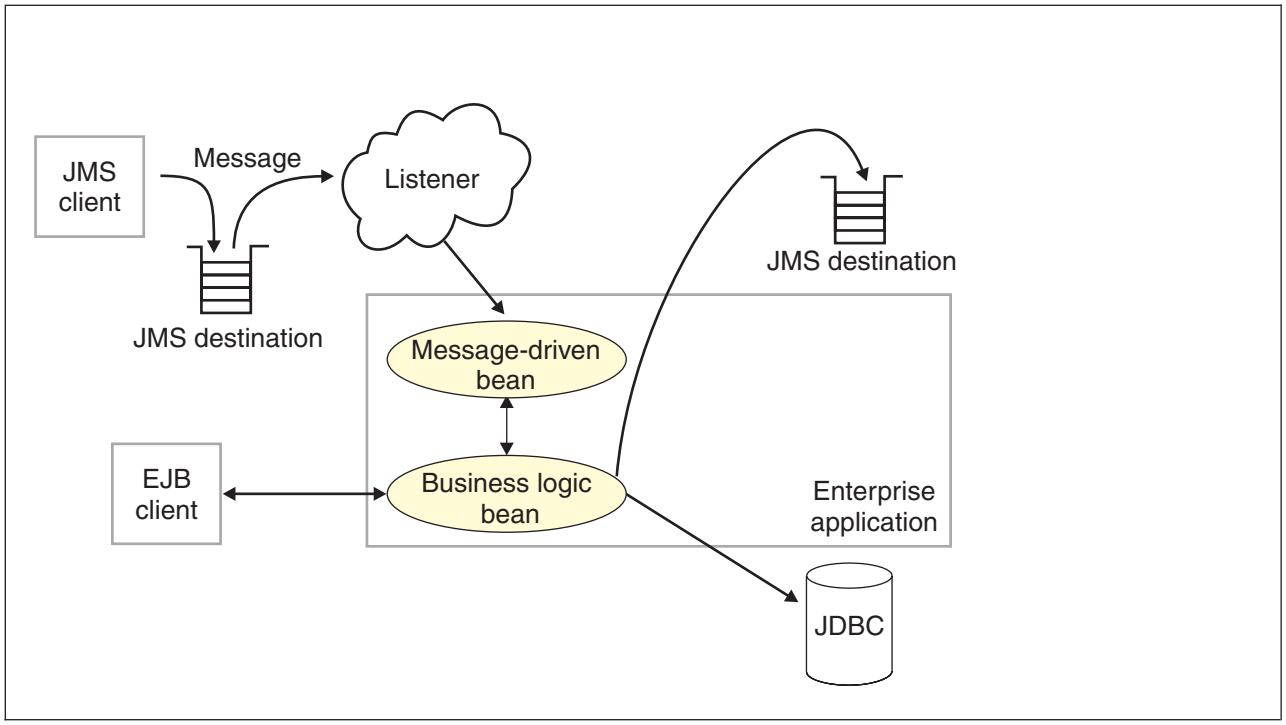


Figure 4. Messaging with message-driven beans. This figure shows an incoming message being passed automatically to the onMessage() method of a message-driven bean that is deployed as a listener for the destination. The message-driven bean processes the message, in this case passing the message on to a business logic bean for business processing. For more information, see the text that accompanies this figure.

Message-driven beans - JCA components

This topic provides an overview of the administrative components that you configure for message-driven beans as listeners on a Java Connector Architecture (JCA) 1.5 resource adapter.

Components for a JCA resource adapter

To handle non-JMS requests inbound to WebSphere Application Server from enterprise information systems, message-driven beans use a Java Connector Architecture (JCA) 1.5 *resource adapter* written by a third party for that purpose.

With a Java Connector Architecture (JCA) 1.5 resource adapter, a message-driven bean acts as a listener on a specific endpoint. In the JCA 1.5 specification, such message-driven beans are commonly called *message endpoints* or simply *endpoints*.

Each application configuring one or more message-driven beans must specify the resource adapter that sends messages to the endpoint. To specify the resource adapter, you configure the message-driven bean to use an activation specification that has been configured by the administrator for the resource adapter.

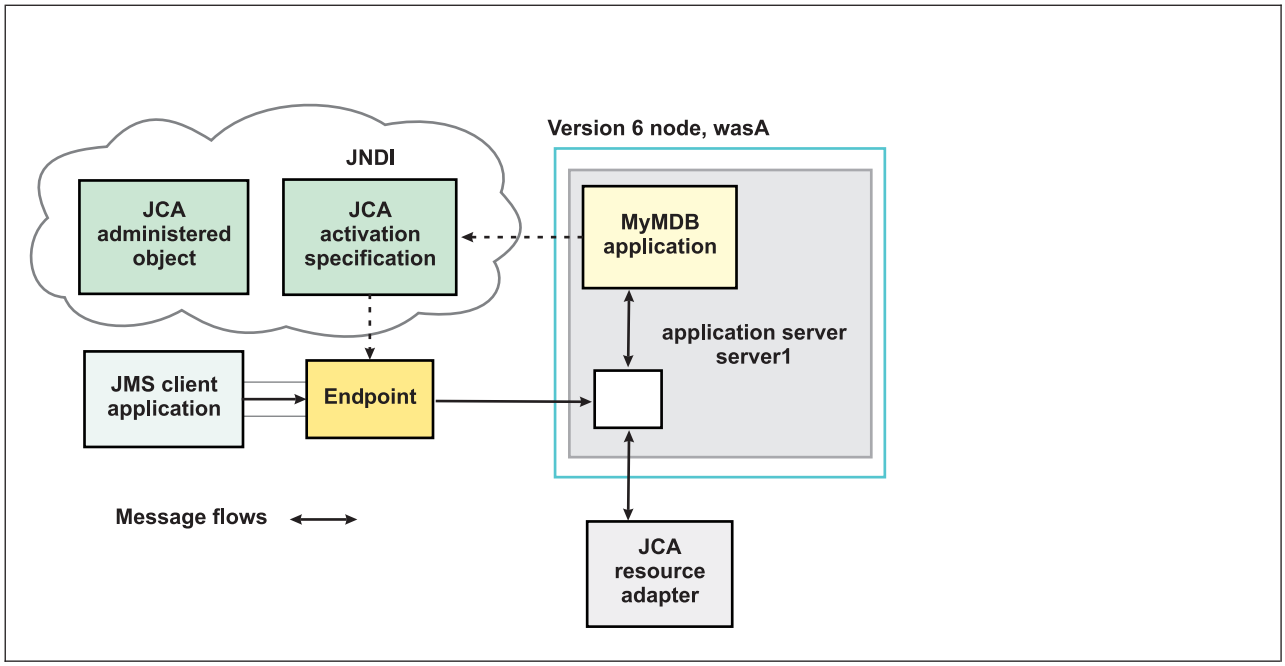


Figure 5. Message-driven bean components for a JCA resource adapter. This figure shows the main components of WebSphere support for message-driven beans for use with an external JCA resource adapter.

The administrator creates a *J2C activation specification* for the appropriate resource adapter to provide information to the deployer about the configuration properties of an endpoint instance (message-driven bean) related to the processing of the inbound messages. Properties specified on an activation specification can be overridden by appropriately named activation-configuration properties in the deployment descriptor of an associated EJB 2.1 message-driven bean.

When a deployed message-driven bean is installed, it is associated with an activation specification for an endpoint. When a message arrives on the endpoint, the message is passed to a new instance of a message-driven bean for processing.

Administered object definitions and classes are provided by a resource adapter when you install it. Using this information, the administrator can create and configure J2C *administered objects* with JNDI names that are then available for applications to use. Some messaging styles may need applications to use special administered objects for sending and synchronously receiving messages (through connection objects using programming interfaces specific to a messaging style). Administered objects can also be used to perform transformations on an asynchronously-received message in a way that is specific to a message provider. Administered objects can be accessed by a component by using either a message destination reference (preferred) or a resource environment reference.

JMS components used with a JCA messaging provider

Message-driven beans that implement the javax.jms.MessageListener interface can be used for JMS messaging. For JMS messaging, message-driven beans can use a JCA-based messaging provider such as the SIB JMS Resource Adapter (which is the default messaging provider listed under JMS providers) that is part of WebSphere Application Server and configure message-driven beans to use a JCA activation specification.

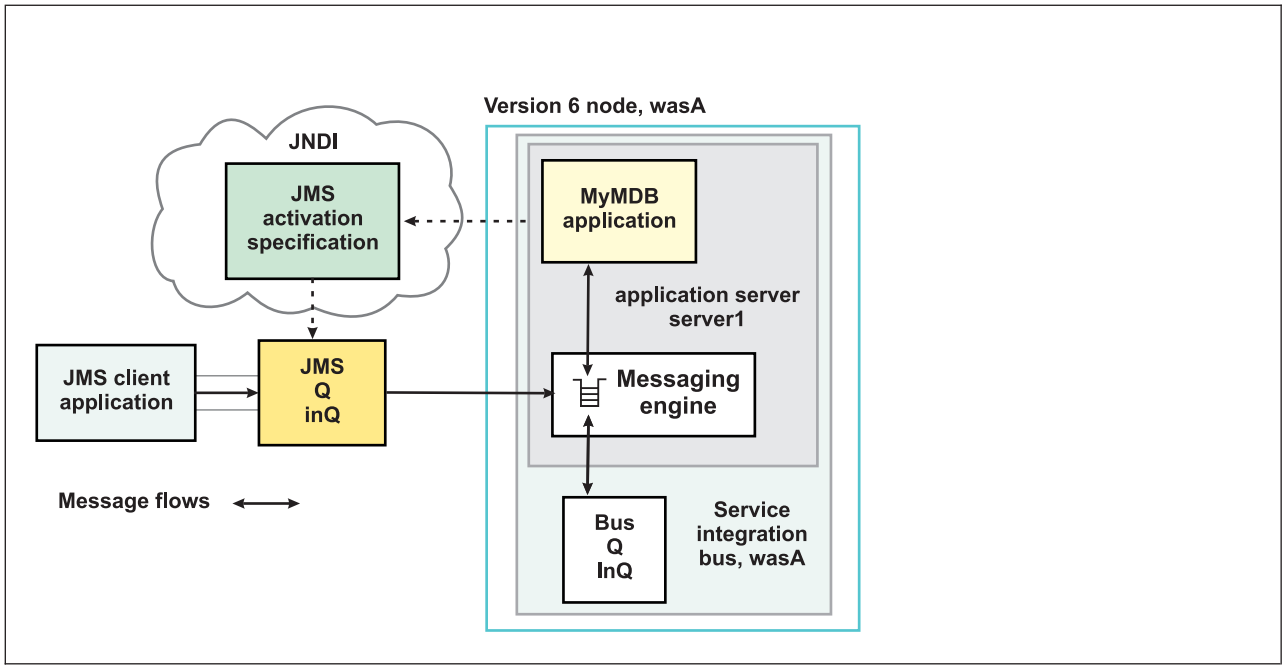


Figure 6. Message-driven bean components for the default messaging provider. This figure shows the main components of WebSphere support for message-driven beans for use with the default messaging provider.

With the SIB JMS Resource Adapter, a message-driven bean acts as a listener on a specific *JMS destination*.

The administrator creates a *JMS activation specification* (which, the WebSphere administrative console shows on the panel **Resources** → **JMS providers** → **Default messaging** → **JMS activation specifications**) to provide information to the deployer about the configuration properties of a message-driven bean related to the processing of the inbound messages. WebSphere provides additional support for JCA activation specifications that are JMS-based and shows the JMS-specific panel rather than the generic JCA activation specification panel. For example, a JMS activation specification specifies the name of the service integration bus to connect to, and includes information about the message acknowledgement modes, message selectors, destination types, and whether or not durable subscriptions are shared across connections with members of a server cluster. Properties specified on an activation specification can be overridden by appropriately named activation-configuration properties in the deployment descriptor of an associated EJB 2.1 message-driven bean.

The administrator also creates other administered objects that configure the JMS destination and the associated resources of a service integration bus that are used to implement messaging with that JMS destination.

For more information about JMS resources and service integration, see:

- Learning about the default messaging provider

J2C activation specification configuration and use

This topic provides an overview about the configuration and use of J2C activation specifications, used in the deployment of message-driven beans for JCA 1.5 resources.

J2C activation specifications are part of the configuration of inbound messaging support that can be part of a JCA 1.5 resource adapter. Each JCA 1.5 resource adapter that supports inbound messaging defines one or more types of message listener in its deployment descriptor (`messagelistener` in the ra.xml). The message listener is the interface that the resource adapter uses to communicate inbound messages to the message endpoint. A message-driven bean (MDB) is a message endpoint and implements one of the message listener interfaces provided by the resource adapter. By allowing multiple types of message

listener, a resource adapter can support a variety of different protocols. For example, the interface javax.jms.MessageListener, is a type of message listener that supports JMS messaging. For each type of message listener that a resource adapter implements, the resource adapter defines an associated activation specification (`activationspec` in the ra.xml). The activation specification is used to set configuration properties for a particular use of the inbound support for the receiving endpoint.

When an application containing a message-driven bean is deployed, the deployer must select a resource adapter that supports the same type of message listener that the message-driven bean implements. As part of the message-driven bean deployment, the deployer needs to specify the properties to set on the J2C activation specification. Later, during application startup, a J2C activation specification instance is created, and these properties are set and used to activate the endpoint (that is, to configure the resource adapter's inbound support for the specific message-driven bean).

Applications with message-driven beans can also specify all, some, or none of the configuration properties needed by the ActivationSpec class, to override those defined by the resource adapter-scoped definition. These properties, specified as activation-config properties in the application's deployment descriptor, are configured when the application is assembled. To change any of these properties requires redeploying the application. These properties are unique to this applications use and are not shared with other message-driven beans. Any properties defined in the application's deployment descriptor take precedence over those defined by the resource adapter-scoped definition. This allows application developers to choose the best defaults for their applications.

WebSphere activation specification optional binding properties

Binding properties that you can specify for activation specifications to be deployed on WebSphere Application Server.

J2C authentication alias

If you provide values for user name and password as custom properties on an activation specification, you may not want to have those values exposed in clear text for security reasons. WebSphere security allows you to securely define an authentication alias for such cases. Configuration of activation specifications, both as an administrative object and during application deployment, enable you to use the authentication alias instead of providing the user name and password.

If you set the authentication alias field, then you should not set the user name and password custom properties fields. Also, authentication alias properties set as part of application deployment take precedence over properties set on an activation specification administrative object.

Only the authentication alias is ever written to file in an unencrypted form, even for purposes of transaction recovery logging. The security service is used to protect the real user name and password.

During application startup, when the activation specification is being initialized as part of endpoint activation, the server uses the authentication alias to retrieve the real user name and password from security then set it on the activation specification instance.

Destination JNDI name

For resource adapters that support JMS you need to associate javax.jms.Destinations with an activation specification, such that the resource adapter can service messages from the JMS destination. In this case, the administrator configures a J2C Administered Object which implements the javax.jms.Destination interface and binds it into JNDI.

You can configure a J2C Administered Object to use an ActivationSpec class that implements a setDestination(javax.jms.Destination) method. In this case, you can specify the destination JNDI name (that is, the JNDI name for the J2C Administered object that implements the javax.jms.Destination).

A destination JNDI name set as part of application deployment take precedence over properties set on an activation specification administrative object.

During application startup, when the activation specification is being initialized as part of endpoint activation, the server uses the destination JNDI name to look up the destination administered object then set it on the activation specification instance.

Message-driven beans - transaction support

Message-driven beans can handle messages on destinations (or endpoints) within the scope of a transaction.

Destination transaction handling

If transaction handling is specified for a destination, the message-driven bean starts a global transaction *before* it reads any incoming message from that destination. When the message-driven bean processing has finished, it commits or rolls back the transaction (using JTA transaction control).

All messages retrieved from a specific destination have the same transactional behavior.

If messages are queued to be sent within a global transaction they are sent when the transaction is committed. If the processing of a message causes the transaction to be rolled back, then the message that caused the bean instance to be invoked is left on the JMS destination.

Inbound resource adapter transaction handling

A message-driven bean can be set up to either have Bean or Container transaction handling. The resource adapter owner must tell the message-driven bean developer how to set up the message-driven bean for transaction handling.

Asynchronous messaging - security considerations

This topic describes considerations that you should be aware of if you want to use security for asynchronous messaging with WebSphere Application Server.

Security for messaging is enabled only when WebSphere Application Server security is enabled.

The user ID and password do not need to be provided by the application. If authentication is successful, then the JMS connection is created; if the authentication fails then the connection request is ended.

When WebSphere Application Server security is enabled, JMS connections made to the JMS provider are authenticated, and access to JMS resources owned by the JMS provider are controlled by access authorizations. Also, all requests to create new connections to the JMS provider must provide a user ID and password for authentication. The user ID and password do not need to be provided by the application. If authentication is successful, then the JMS connection is created; if the authentication fails then the connection request is ended.

Standard J2C authentication is used for a request to create a new connection to the JMS provider. If your resource authentication (res-auth) is set to Application, set the alias in the Component-managed Authentication Alias. If the application that tries to create a connection to the JMS provider specifies a user ID and password, those values are used to authenticate the creation request. If the application does not specify a user ID and password, the values defined by the Component-managed Authentication Alias are used. If the connection factory is not configured with a Component-managed Authentication Alias, then you receive a runtime JMS exception when an attempt is made to connect to the JMS provider.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with the version 5 default messaging provider or WebSphere MQ. For example, the default Windows NT user ID, **Administrator**, is not valid for use, because it contains 13 characters.

Therefore, an authentication alias for a WebSphere JMS provider or WebSphere MQ connection factory must specify a user ID no longer than 12 characters.

2. If you want to use Bindings transport mode for JMS connections to WebSphere MQ, you set the property **Transport type**=BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager` error.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Authorization to access messages stored by the default messaging provider is controlled by authorization to access the service integration bus destinations on which the messages are stored. For information about authorizing permissions for individual bus destinations, see Administering destination permissions.

Messaging: Resources for learning

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

- Sun's Java Message Service (JMS) specification documentation.
  Provides details about the Java Message Service (JMS).
- Sun's J2EE Connector Architecture specification (http://java.sun.com/j2ee/connector/download.html).
  Provides details about inbound messaging processing using the J2EE Connector architecture.
- J2EE specification
  Provides details about the J2EE specification, including messaging considerations.
- WebSphere MQ Using Java.
  Provides information about using JMS with WebSphere MQ as a messaging provider.
- http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html
  Provides WebSphere MQ messaging platform-specific books.
- WebSphere MQ Event Broker Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#eventb
  Provides books about WebSphere MQ Event Broker as a publish/subscribe messaging broker.
- WebSphere MQ Integrator Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#integrator
  Provides books about WebSphere MQ Integrator as a publish/subscribe messaging broker.
- IBM Publications Center
  This Web site provides a wide range of IBM publications, including publications about messaging products.

Installing and configuring a JMS provider

This topic describes the different ways that you can use JMS providers with WebSphere Application Server. A JMS provider enables use of the Java Message Service (JMS) and other message resources in WebSphere Application Server.

IBM WebSphere Application Server supports asynchronous messaging through the use of a JMS provider and its related messaging system. JMS providers must conform to the JMS specification version 1.1. To use message-driven beans the JMS provider must support the optional Application Server Facility (ASF) function defined within that specification, or support an inbound resource adapter as defined in the JCA specification version 1.5.

The service integration technologies of IBM WebSphere Application Server can act as a messaging system when you have configured a service integration bus that is accessed through the default messaging provider. This support is installed as part of WebSphere Application Server, administered through the administrative console, and is fully integrated with the WebSphere Application Server runtime.

WebSphere Application Server also includes support for the following JMS providers:

WebSphere MQ
Provided for use with supported versions of WebSphere MQ.

Generic
Provided for use with any 3rd party messaging system. If you want to use message-driven beans, the messaging system must support ASF.

For more information about the support for JMS providers, see “JMS providers” on page 811.

For more information about installing and using JMS providers, see the following topics:

- Installing the default messaging provider
- Using WebSphere MQ as a JMS provider. Installing WebSphere MQ as a JMS provider.

  Note:
  - You can install WebSphere MQ in addition to the default messaging provider. The preferred solution for publish/subscribe messaging with WebSphere MQ as a JMS provider is a full message broker such as WebSphere MQ Event Broker.
  - If you install WebSphere MQ as a JMS provider, you can use the WebSphere administrative console to administer the JMS resources provided by WebSphere MQ, such as queue connection factories. However, you cannot administer MQ security, which is administered through WebSphere MQ.

  For more information about scenarios and considerations for using WebSphere MQ with IBM WebSphere Application Server, see the White Papers and Red books provided by WebSphere MQ; for example, through the WebSphere MQ library Web page at http://www-3.ibm.com/software/ts/mqseries/library/
- Installing another JMS provider, which must conform to the JMS specification and, to use message-driven beans, support the ASF function. If you want to use a JMS provider other than the default messaging provider or WebSphere MQ, you should complete the following steps:
  1. Installing and configuring the JMS provider and its resources by using the tools and information provided with the product.
  2. Defining the JMS provider to WebSphere Application Server as a generic messaging provider.

  Note: You can use the WebSphere administrative console to administer JMS connection factories and destinations (within WebSphere Application Server) for a generic provider, but cannot administer the JMS provider or its resources outside of WebSphere Application Server.

Installing the default messaging provider

Use this task to install the default messaging provider of IBM WebSphere Application Server.

The default messaging provider is installed as a fully-integrated component of WebSphere Application Server, and needs no separate installation steps. However, ensure that there is enough space in the file systems where you want to store messaging data.

You can use the WebSphere administrative console to Using the default messaging provider for the default messaging provider.

For more information about the default messaging provider, see Using the default messaging provider .

JMS providers collection

A JMS provider enables messaging based on the Java Message Service (JMS). It provides J2EE connection factories to create connections for JMS destinations.

In the administrative console page, to view this page click **Resources** → **JMS** → **JMS providers**

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS providers that are available to WebSphere applications. For each JMS provider in the list, the entry indicates the *scope* level at which JMS resource definitions are visible to applications. You can create the same type of JMS provider at different Scope settings, to offer JMS resources at different levels of visibility to applications.

If you want to manage existing JMS resource definitions, or create a new JMS resource definition, you can select the name of one of the JMS providers in the list.

If you want to define your own JMS provider, other than the default messaging provider or WebSphere MQ, select the Scope setting at which JMS resource definitions are to be visible for that provider, then click **New**.

General properties

Name The name of the JMS provider.

Description
A description of the JMS provider.

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Select JMS resource provider

Select the provider with which to create the {0}. The following providers support the selected resource type and are available at the selected scope. The variable, *{0}*, indicates the type of JMS resource that you are creating.

You select the scope setting on an earlier page. The choice of JMS providers depends on the scope that you selected. You might see a choice like the following list:

- Default messaging provider.

  Select this option if you want to the type of JMS resource to be provided by a service integration bus, as part of WebSphere Application Server.
- My JMSprovider

  Select this option if you want the type of JMS resource to be provided by your own JMS provider; not the default messaging provider or WebSphere MQ. You assign the name, in this example “My

JMSprovider", when you define the JMS provider to WebSphere Application Server. You must have installed and configured your own JMS provider before applications can use the JMS resources.

- WebSphere MQ messaging provider

  Select this option if you want the type of JMS resource to be provided by WebSphere MQ. You must have installed and configured WebSphere MQ before applications can use the JMS resources.

Activation specification collection

A JMS activation specification is associated with one or more message-driven beans and provides configuration necessary for them to receive messages.

In the administrative console page, to view this page click **Resources** → **JMS** → **Activation specification**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS activation specifications that are available to WebSphere applications at the scope indicated by the **Scope** field.

Use a JMS activation specification if you want to use a message-driven bean as a Java Connector Architecture (JCA) 1.5 resource, to act as a listener on the default messaging provider.

General properties

Name The name of the activation specification.

Provider
This JMS resource is for the *Default messaging provider*, provided by service integration technologies as part of WebSphere Application Server.

Description
A description of the activation specification.

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Connection factory collection

Use this page to create connections to the associated JMS provider for JMS destinations.

In the administrative console page, to view this page click **Resources** → **JMS** → **Connection factory**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS connection factories that are available to WebSphere applications at the scope indicated by the **Scope** field.

A JMS connection factory is used to create connections to JMS destinations. When an application needs a JMS connection, an instance can be created by the factory of the JMS provider named in the Provider column of the list.

This type of connection factory is for applications that use the JMS 1.1 domain-independent interfaces (referred to as the “common interfaces” in the JMS specification).

This type of JMS connection factory can also be used by the domain-specific (queue and topic) interfaces, as used in JMS 1.0.2, so applications can still use those interfaces without the need for you to create a domain-specific connection factory, such as a queue connection factory.

General properties

Name The name of the connection factory.

Provider
Specifies a JMS provider, which enables asynchronous messaging based on the Java Message Service (JMS). It provides J2EE connection factories to create connections for specific JMS queue or topic destinations. JMS provider administrative objects are used to manage JMS resources for the associated JMS provider.

Description
A description of the connection factory.

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Queue connection factory collection

A queue connection factory is used to create connections to the associated JMS provider of the JMS queue destinations, for point-to-point messaging.

In the administrative console page, to view this page click **Resources** → **JMS** → **Queue connection factory**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS queue connection factories that are available to WebSphere applications at the scope indicated by the **Scope** field.

A JMS connection factory is used to create connections to JMS destinations. When an application needs a JMS connection, an instance can be created by the factory for the JMS provider that is named in the Provider column of the list.

This type of connection factory is for applications that use the JMS 1.0.2 queue-specific interfaces.

General properties

Name The name of the queue connection factory.

Provider
The type of JMS provider that provides the JMS resource. For example, for *Default messaging provider* resources are provided by service integration technologies as part of WebSphere Application Server; for *WebSphere MQ messaging provider* resources are provided by a separate WebSphere MQ installation.

Description
A description of the queue connection factory.

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Queue collection

A JMS queue is used as a destination for point-to-point messaging.

In the administrative console page, to view this page click **Resources** → **JMS** → **Queue**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS queue destinations that are available to WebSphere applications at the scope indicated by the **Scope** field.

Use topic destination administrative objects to manage JMS queues for the JMS provider that is named in the Provider column of the list. Connections to the queue are created by a connection factory (or queue connection factory) for that JMS provider.

General properties

Name The name of the queue.

Provider
Specifies a JMS provider, which enables asynchronous messaging based on the Java Message Service (JMS). It provides J2EE connection factories to create connections for specific JMS queue or topic destinations. JMS provider administrative objects are used to manage JMS resources for the associated JMS provider.

Description
A description of the queue.

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Topic connection factory collection

A topic connection factory is used to create connections to the associated JMS provider of JMS topic destinations, for publish and subscribe messaging.

In the administrative console page, to view this page click **Resources** → **JMS** → **Topic connection factory**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS topic connection factories that are available to WebSphere applications at the scope indicated by the **Scope** field.

A JMS connection factory is used to create connections to JMS destinations. When an application needs a JMS connection, an instance can be created by the factory for the JMS provider that is named in the Provider column of the list.

This type of connection factory is for applications that use the JMS 1.0.2 topic-specific interfaces.

General properties

Name The name of the topic connection factory.

Provider
The type of JMS provider that provides the JMS resource. For example, for *Default messaging provider* resources are provided by service integration technologies as part of WebSphere Application Server; for *WebSphere MQ messaging provider* resources are provided by a separate WebSphere MQ installation.

Description
A description of the topic connection factory.

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Topic collection

A JMS topic is used as a destination for publish/subscribe messaging.

In the administrative console page, to view this page click **Resources** → **JMS** → **Topic**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS topic destinations that are available to WebSphere applications at the scope indicated by the **Scope** field.

Use topic destination administrative objects to manage JMS topics for the JMS provider that is named in the Provider column of the list. Connections to the topic are created by a connection factory (or topic connection factory) for that JMS provider.

General properties

Name The name of the topic.

Provider
The type of JMS provider that provides the JMS resource. For example, for *Default messaging provider* resources are provided by service integration technologies as part of WebSphere Application Server; for *WebSphere MQ messaging provider* resources are provided by a separate WebSphere MQ installation.

Description
A description of the topic.

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Configuring messaging with scripting

Use these topics to learn about configuring messaging with scripting and the wsadmin tool.

This topic contains the following tasks:

Configuring the message listener service using scripting

Use scripting to configure the message listener service.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure the message listener service for an application server:

1. Identify the application server and assign it to the server variable:
   - Using Jacl:

     ```
     set server [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```

   - Using Jython:

     ```
     server = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print server
     ```

   Example output:

   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
   ```

2. Identify the message listener service belonging to the server and assign it to the mls variable:
   - Using Jacl:

     ```
     set mls [$AdminConfig list MessageListenerService $server]
     ```

   - Using Jython:

     ```
     mls = AdminConfig.list('MessageListenerService', server)
     print mls
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode/servers/server1|server.xml#MessageListenerService_1)
   ```

3. Modify various attributes with one of the following examples:
   - This example command changes the thread pool attributes:
     - Using Jacl:

       ```
       $AdminConfig modify $mls {{threadPool {{inactivityTimeout 4000}
       {isGrowable true} {maximumSize 100} {minimumSize 25}}}}
       ```

     - Using Jython:

       ```
       AdminConfig.modify(mls, [['threadPool', [['inactivityTimeout', 4000],
       ['isGrowable',  'true'], ['maximumSize', 100], ['minimumSize', 25]]]])
       ```

   - This example modifies the property of the first listener port:
     - Using Jacl:

       ```
       set lports [$AdminConfig showAttribute $mls listenerPorts]
       set lport [lindex $lports 0]
       $AdminConfig modify $lport {{maxRetries 2}}
       ```

     - Using Jython:

       ```
       lports = AdminConfig.showAttribute(mls, 'listenerPorts')
       cleanLports = lports[1:len(lports)-1]
       lport = cleanLports.split(" ")[0]
       AdminConfig.modify(lport, [['maxRetries', 2]])
       ```

   - This example adds a listener port:
     - Using Jacl:

       ```
       set new [$AdminConfig create ListenerPort $mls {{name my}
       {destinationJNDIName di} {connectionFactoryJNDIName jndi/fs}}]
       $AdminConfig create StateManageable $new {{initialState START}}
       ```

     - Using Jython:

       ```
       new = AdminConfig.create('ListenerPort', mls, [['name', 'my'],
       ['destinationJNDIName', 'di'], ['connectionFactoryJNDIName', 'jndi/fsi']])
       print new
       print AdminConfig.create('StateManageable', new, [['initialState', 'START']])
       ```

     Example output:

     ```
     my(cells/mycell/nodes/mynode/servers/server1:server.xml#ListenerPort_1079471940692)
     (cells/mycell/nodes/mynode/servers/server1:server.xml#StateManageable_107947182623)
     ```

4. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

5. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new JMS providers using scripting

You can use the wsadmin tool and scripting to configure a new JMS provider.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new JMS provider:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```
   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```
2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required JMSProvider
     ```
   - Using Jython:

     ```
     print AdminConfig.required('JMSProvider')
     ```

   Example output:

   ```
   Attribute        Type
   name     String
   externalInitialContextFactory    String
   externalProviderURL               String
   ```
3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name JMSP1]
     set extICF [list externalInitialContextFactory
     "Put the external initial context factory here"]
     set extPURL [list externalProviderURL "Put the external provider URL here"]
     set jmspAttrs [list $name $extICF $extPURL]
     ```
   - Using Jython:

     ```
     name = ['name', 'JMSP1']
     extICF = ['externalInitialContextFactory',
     "Put the external initial context factory here"]
     extPURL = ['externalProviderURL', "Put the external provider URL here"]
     jmspAttrs = [name, extICF, extPURL]
     print jmspAttrs
     ```

   Example output:

   ```
   {name JMSP1} {externalInitialContextFactory {Put the external
   initial context factory here }} {externalProviderURL
   {Put the external provider URL here}}
   ```
4. Create the JMS provider:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig create JMSProvider $node $jmspAttrs]
     ```
   - Using Jython:

     ```
     newjmsp = AdminConfig.create('JMSProvider', node, jmspAttrs)
     print newjmsp
     ```

Example output:

```
JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new JMS destinations using scripting

You can use scripting and the wsadmin tool to configure a new JMS destination.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new JMS destination:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:myNode/JMSProvider:JMSP1]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required GenericJMSDestination
     ```

   - Using Jython:

     ```
     print AdminConfig.required('GenericJMSDestination')
     ```

   Example output:

   ```
   Attribute       Type
   name       String
   jndiName    String
   externalJNDIName  String
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name JMSD1]
     set jndi [list jndiName jms/JMSDestination1]
     set extJndi [list externalJNDIName jms/extJMSD1]
     set jmsdAttrs [list $name $jndi $extJndi]
     ```

   - Using Jython:

     ```
     name = ['name', 'JMSD1']
     jndi = ['jndiName', 'jms/JMSDestination1']
     extJndi = ['externalJNDIName', 'jms/extJMSD1']
     jmsdAttrs = [name, jndi, extJndi]
     print jmsdAttrs
     ```

   Example output:

   ```
   {name JMSD1} {jndiName jms/JMSDestination1} {externalJNDIName jms/extJMSD1}
   ```

4. Create generic JMS destination:
   - Using Jacl:

     ```
     $AdminConfig create GenericJMSDestination $newjmsp  $jmsdAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('GenericJMSDestination', newjmsp,  jmsdAttrs)
     ```

Example output:

```
JMSD1(cells/mycell/nodes/mynode|resources.xml#GenericJMSDestination_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new JMS connections using scripting

Use scripting and the wsadmin tool to configure a new JMS connection.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new JMS connection:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:myNode/JMSProvider:JMSP1]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required GenericJMSConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('GenericJMSConnectionFactory')
     ```

   Example output:

   ```
   Attribute       Type
   name       String
   jndiName    String
   externalJNDIName  String
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name JMSCF1]
     set jndi [list jndiName jms/JMSConnFact1]
     set extJndi [list externalJNDIName jms/extJMSCF1]
     set jmscfAttrs [list $name $jndi $extJndi]
     ```

     Example output:

     ```
     {name JMSCF1} {jndiName jms/JMSConnFact1} {externalJNDIName jms/extJMSCF1}
     ```

   - Using Jython:

     ```
     name = ['name', 'JMSCF1']
     jndi = ['jndiName', 'jms/JMSConnFact1']
     extJndi = ['externalJNDIName', 'jms/extJMSCF1']
     jmscfAttrs = [name, jndi, extJndi]
     print jmscfAttrs
     ```

     Example output:

     ```
     [[name, JMSCF1], [jndiName, jms/JMSConnFact1], [externalJNDIName, jms/extJMSCF1]]
     ```

4. Create generic JMS connection factory:
   - Using Jacl:

     ```
     $AdminConfig create GenericJMSConnectionFactory $newjmsp $jmscfAttrs
     ```

- Using Jython:

```
print AdminConfig.create('GenericJMSConnectionFactory', newjmsp, jmscfAttrs)
```

Example output:

```
JMSCF1(cells/mycell/nodes/mynode|resources.xml#GenericJMSConnectionFactory_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new WebSphere queue connection factories using scripting

You can use scripting and the wsadmin tool to configure new queue connection factories in WebSphere Application Server.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new WebSphere queue connection factory:

1. Identify the parent ID:
   - Using Jacl:

```
set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
```

   - Using Jython:

```
newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1/')
print newjmsp
```

   Example output:

```
JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
```

2. Get required attributes:
   - Using Jacl:

```
$AdminConfig required WASQueueConnectionFactory
```

   - Using Jython:

```
print AdminConfig.required('WASQueueConnectionFactory')
```

   Example output:

```
Attribute       Type
name       String
jndiName    String
```

3. Set up required attributes:
   - Using Jacl:

```
set name [list name WASQCF]
set jndi [list jndiName jms/WASQCF]
set mqcfAttrs [list $name $jndi]
```

     Example output:

```
{name WASQCF} {jndiName jms/WASQCF}
```

   - Using Jython:

```
name = ['name', 'WASQCF']
jndi = ['jndiName', 'jms/WASQCF']
mqcfAttrs = [name, jndi]
print mqcfAttrs
```

     Example output:

```
[[name, WASQCF], [jndiName, jms/WASQCF]]
```

4. Create was queue connection factories:

- Using Jacl:

  ```
  $AdminConfig create WASQueueConnectionFactory $newjmsp $mqcfAttrs
  ```

- Using Jython:

  ```
  print AdminConfig.create('WASQueueConnectionFactory', newjmsp, mqcfAttrs)
  ```

  Example output:

  ```
  WASQCF(cells/mycell/nodes/mynode|resources.xml#WASQueueConnectionFactory_1)
  ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new WebSphere topic connection factories using scripting

Use scripting and the wsadmin tool to configure new WebSphere topic connection factories.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new WebSphere topic connection factory:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1/')
     print newjmsp
     ```

     Example output:

     ```
     JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
     ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASTopicConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WASTopicConnectionFactory')
     ```

     Example output:

     ```
     Attribute            Type
     name                 String
     jndiName             String
     port                 ENUM(DIRECT, QUEUED)
     ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name WASTCF]
     set jndi [list jndiName jms/WASTCF]
     set port [list port QUEUED]
     set mtcfAttrs [list $name $jndi $port]
     ```

     Example output:

     ```
     {name WASTCF} {jndiName jms/WASTCF} {port QUEUED}
     ```

   - Using Jython:

     ```
     name = ['name', 'WASTCF']
     jndi = ['jndiName', 'jms/WASTCF']
     port = ['port', 'QUEUED']
     mtcfAttrs = [name, jndi, port]
     print mtcfAttrs
     ```

     Example output:

```
[[name, WASTCF], [jndiName, jms/WASTCF], [port, QUEUED]]
```

4. Create was topic connection factories:
   - Using Jacl:
     ```
     $AdminConfig create WASTopicConnectionFactory $newjmsp $mtcfAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('WASTopicConnectionFactory', newjmsp, mtcfAttrs)
     ```

   Example output:
   ```
   WASTCF(cells/mycell/nodes/mynode|resources.xml#WASTopicConnectionFactory_1)
   ```
5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new WebSphere queues using scripting

You can use scripting to configure a new WebSphere queue.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new WebSphere queue:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
     ```
   - Using Jython:
     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1/')
     print newjmsp
     ```

   Example output:
   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required WASQueue
     ```
   - Using Jython:
     ```
     print AdminConfig.required('WASQueue')
     ```

   Example output:
   ```
   Attribute       Type
   name       String
   jndiName    String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set name [list name WASQ1]
     set jndi [list jndiName jms/WASQ1]
     set wqAttrs [list $name $jndi]
     ```

     Example output:
     ```
     {name WASQ1} {jndiName jms/WASQ1}
     ```
   - Using Jython:
     ```
     name = ['name', 'WASQ1']
     jndi = ['jndiName', 'jms/WASQ1']
     wqAttrs = [name, jndi]
     print wqAttrs
     ```

     Example output:

```
[[name, WASQ1], [jndiName, jms/WASQ1]]
```

4. Create was queue:
   - Using Jacl:

     ```
     $AdminConfig create WASQueue $newjmsp $wqAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('WASQueue', newjmsp, wqAttrs)
     ```

   Example output:

   ```
   WASQ1(cells/mycell/nodes/mynode|resources.xml#WASQueue_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new WebSphere topics using scripting

You can configure new WebSphere topics using the wsadmin tool and scripting.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new WebSphere topic:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1/')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASTopic
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WASTopic')
     ```

   Example output:

   ```
   Attribute       Type
   name       String
   jndiName    String
   topic        String
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name WAST1]
     set jndi [list jndiName jms/WAST1]
     set topic [list topic "Put your topic here"]
     set wtAttrs [list $name $jndi $topic]
     ```

     Example output:

     ```
     {name WAST1} {jndiName jms/WAST1} {topic {Put your topic here}}
     ```

   - Using Jython:

```
name = ['name', 'WAST1']
jndi = ['jndiName', 'jms/WAST1']
topic = ['topic', "Put your topic here"]
wtAttrs = [name, jndi, topic]
print wtAttrs
```

Example output:

```
[[name, WAST1], [jndiName, jms/WAST1], [topic, "Put your topic here"]]
```

4. Create was topic:
   - Using Jacl:

     ```
     $AdminConfig create WASTopic $newjmsp $wtAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('WASTopic', newjmsp, wtAttrs)
     ```

   Example output:

   ```
   WAST1(cells/mycell/nodes/mynode|resources.xml#WASTopic_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new MQ connection factories using scripting

You can use scripting to configure a new MQ connection factory.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new MQ connection factory:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:WebSphere MQ JMS Provider/]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere MQ JMS Provider')
     print newjmsp
     ```

   Example output:

   ```
   WebSphere MQ JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_mqprovider)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required MQConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('MQConnectionFactory')
     ```

   Example output:

   ```
   ttribute       Type
   name           String
   jndiName          String
   connectionPool     ConnectionPool
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name MQCF]
     set jndi [list jndiName jms/MQCF]
     set mqcfAttrs [list $name $jndi]
     ```

     Example output:

     ```
     {name MQCF} {jndiName jms/MQCF}
     ```

- Using Jython:

```
name = ['name', 'MQCF']
jndi = ['jndiName', 'jms/MQCF']
mqcfAttrs = [name, jndi]
print mqcfAttrs
```

Example output:

```
[[name, MQCF], [jndiName, jms/MQCF]]
```

4. Set up a template:
   - Using Jacl:

```
set template [lindex [$AdminConfig listTemplates MQConnectionFactory] 0]
```

   - Using Jython:

```
import java
lineseparator = java.lang.System.getProperty('line.separator')
template = AdminConfig.listTemplates('MQConnectionFactory').split(lineseparator)[0]
print template
```

   Example output:

```
Example non-XA WMQ ConnectionFactory(templates/
system:JMS-resource-provider-templates.xml
#MQConnectionFactory_3)
```

5. Create MQ connection factory:
   - Using Jacl:

```
$AdminConfig createUsingTemplate MQConnectionFactory
$newjmsp $mqcfAttrs $template
```

   - Using Jython:

```
print AdminConfig.createUsingTemplate('MQConnectionFactory',
newjmsp, mqcfAttrs, template)
```

   Example output:

```
MQCF(cells/mycell/nodes/mynode:resources.xml#MQConnectionFactory_1)
```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new MQ queue connection factories using scripting

You can use scripting to configure a new MQ queue connection factory.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new MQ queue connection factory:

1. Identify the parent ID:
   - Using Jacl:

```
set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
```

   - Using Jython:

```
newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
print newjmsp
```

   Example output:

```
JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
```

2. Get required attributes:
   - Using Jacl:

```
$AdminConfig required MQQueueConnectionFactory
```

- Using Jython:

```
print AdminConfig.required('MQQueueConnectionFactory')
```

Example output:

```
Attribute       Type
name       String
jndiName    String
```

3. Set up required attributes:
   - Using Jacl:

```
set name [list name MQQCF]
set jndi [list jndiName jms/MQQCF]
set mqqcfAttrs [list $name $jndi]
```

Example output:

```
{name MQQCF} {jndiName jms/MQQCF}
```

   - Using Jython:

```
name = ['name', 'MQQCF']
jndi = ['jndiName', 'jms/MQQCF']
mqqcfAttrs = [name, jndi]
print mqqcfAttrs
```

Example output:

```
[[name, MQQCF], [jndiName, jms/MQQCF]]
```

4. Set up a template:
   - Using Jacl:

```
set template [lindex [$AdminConfig listTemplates MQQueueConnectionFactory] 0]
```

   - Using Jython:

```
import java
lineseparator = java.lang.System.getProperty('line.separator')
template = AdminConfig.listTemplates('MQQueueConnectionFactory').split(lineseparator)[0]
print template
```

Example output:

```
Example non-XA WMQ QueueConnectionFactory(templates/
system:JMS-resource-provider-templates.xml
#MQQueueConnectionFactory_3)
```

5. Create MQ queue connection factory:
   - Using Jacl:

```
$AdminConfig createUsingTemplate MQQueueConnectionFactory
$newjmsp $mqqcfAttrs $template
```

   - Using Jython:

```
print AdminConfig.createUsingTemplate('MQQueueConnectionFactory',
newjmsp, mqqcfAttrs, template)
```

Example output:

```
MQQCF(cells/mycell/nodes/mynode:resources.xml#MQQueueConnectionFactory_1)
```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new MQ topic connection factories using scripting

Use scripting and the wsadmin tool to configure a new MQ topic connection factory.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new MQ topic connection factory:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
     ```
   - Using Jython:
     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:
   ```
   JMSP1(cells/mycell/nodes/mynode:resources.xml#JMSProvider_1)
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required MQTopicConnectionFactory
     ```
   - Using Jython:
     ```
     print AdminConfig.required('MQTopicConnectionFactory')
     ```

   Example output:
   ```
   Attribute       Type
   name       String
   jndiName    String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set name [list name MQTCF]
     set jndi [list jndiName jms/MQTCF]
     set mqtcfAttrs [list $name $jndi]
     ```

     Example output:
     ```
     {name MQTCF} {jndiName jms/MQTCF}
     ```
   - Using Jython:
     ```
     name = ['name', 'MQTCF']
     jndi = ['jndiName', 'jms/MQTCF']
     mqtcfAttrs = [name, jndi]
     print mqtcfAttrs
     ```

     Example output:
     ```
     [[name, MQTCF], [jndiName, jms/MQTCF]]
     ```
4. Set up a template:
   - Using Jacl:
     ```
     set template [lindex [$AdminConfig listTemplates MQTopicConnectionFactory] 0]
     ```
   - Using Jython:
     ```
     import java
     lineseparator = java.lang.System.getProperty('line.separator')
     template = AdminConfig.listTemplates('MQTopicConnectionFactory').split(lineseparator)[0]
     print template
     ```

   Example output:
   ```
   Example non-XA WMQ TopicConnectionFactory(templates/system:
   JMS-resource-provider-templates.xml
   #MQTopicConnectionFactory_5)
   ```
5. Create mq topic connection factory:
   - Using Jacl:
     ```
     $AdminConfig create MQTopicConnectionFactory $newjmsp $mqtcfAttrs $template
     ```
   - Using Jython:
     ```
     print AdminConfig.create('MQTopicConnectionFactory', newjmsp, mqtcfAttrs, template)
     ```

   Example output:

```
MQTCF(cells/mycell/nodes/mynode:resources.xml#MQTopicConnectionFactory_1)
```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new MQ queues using scripting

You can use scripting and the wsadmin tool to configure a new MQ queue.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new MQ queue:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

     Example output:

     ```
     JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
     ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required MQQueue
     ```

   - Using Jython:

     ```
     print AdminConfig.required('MQQueue')
     ```

     Example output:

     ```
     Attribute       Type
     name       String
     jndiName    String
     baseQueueName  String
     ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name MQQ]
     set jndi [list jndiName jms/MQQ]
     set baseQN [list baseQueueName "Put the base queue name here"]
     set mqqAttrs [list $name $jndi $baseQN]
     ```

     Example output:

     ```
     {name MQQ} {jndiName jms/MQQ} {baseQueueName {Put the base queue name here}}
     ```

   - Using Jython:

     ```
     name = ['name', 'MQQ']
     jndi = ['jndiName', 'jms/MQQ']
     baseQN = ['baseQueueName', "Put the base queue name here"]
     mqqAttrs = [name, jndi, baseQN]
     print mqqAttrs
     ```

     Example output:

     ```
     [[name, MQQ], [jndiName, jms/MQQ], [baseQueueName, "Put the base queue name here"]]
     ```

4. Set up a template:
   - Using Jacl:

     ```
     set template [lindex [$AdminConfig listTemplates MQQueue] 0]
     ```

   - Using Jython:

```
import java
lineseparator = java.lang.System.getProperty('line.separator')
template = AdminConfig.listTemplates('MQQueue').split(lineseparator)[0]
print template
```

Example output:

```
Example.JMS.WMQ.Q1(templates/system:JMS-resource-provider-
templates.xml#MQQueue_1)
```

5. Create MQ queue factory:
   - Using Jacl:

     ```
     $AdminConfig createUsingTemplate MQQueue $newjmsp $mqqAttrs $template
     ```

   - Using Jython:

     ```
     print AdminConfig.createUsingTemplate('MQQueue', newjmsp, mqqAttrs, template)
     ```

   Example output:

   ```
   MQQ(cells/mycell/nodes/mynode|resources.xml#MQQueue_1)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new MQ topics using scripting

You can use scripting to configure a new MQ topic.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new MQ topic:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required MQTopic
     ```

   - Using Jython:

     ```
     print AdminConfig.required('MQTopic')
     ```

   Example output:

   ```
   Attribute            Type
   name                 String
   jndiName             String
   baseTopicName        String
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name MQT]
     set jndi [list jndiName jms/MQT]
     set baseTN [list baseTopicName "Put the base topic name here"]
     set mqtAttrs [list $name $jndi $baseTN]
     ```

   Example output:

```
{name MQT} {jndiName jms/MQT} {baseTopicName {Put the base topic name here}}
```

- Using Jython:

```
name = ['name', 'MQT']
jndi = ['jndiName', 'jms/MQT']
baseTN = ['baseTopicName', "Put the base topic name here"]
mqtAttrs = [name, jndi, baseTN]
print mqtAttrs
```

Example output:

```
[[name, MQT], [jndiName, jms/MQT], [baseTopicName, "Put the base topic name here"]]
```

4. Create MQ topic factory:
   - Using Jacl:

     ```
     $AdminConfig create MQTopic $newjmsp $mqtAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('MQTopic', newjmsp, mqtAttrs)
     ```

   Example output:

   ```
   MQT(cells/mycell/nodes/mynode|resources.xml#MQTopic_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Commands for the JCA management group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the JCA management group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:

copyResourceAdapter	Use the **copyResourceAdapter** command to create a Java 2 Connector (J2C) resource adapter under the scope that you specify.	J2C Resource Adapter object ID	• Parameters: **- name** Indicates the name of the new J2C resource adapter. This parameter is required. **- scope** Indicates the scope object ID. This parameter is required. **- useDeepCopy** If you set this parameter to `true`, all of the J2C connection factory, J2C activation specification, and J2C administrative objects will be copied to the new J2C resource adapter (deep copy). If you set this parameter to `false`, the objects are not created (shallow copy). The default is `false`. • Returns: J2C resource adapter object ID	**Batch mode example usage:** • Using Jacl: `$AdminTask copyResourceAdapter $ra [subst {-name newRA -scope $scope}]` • Using Jython string: `AdminTask.copyResourceAdapter (ra, '[-name newRA -scope scope]')` • Using Jython list: `AdminTask.copyResourceAdapter (ra, ['-name', 'newRA', '-scope', 'scope'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask copyResourceAdapter {-interactive}` • Using Jython string: `AdminTask.copyResourceAdapter ('[-interactive]')` • Using Jython list: `AdminTask.copyResourceAdapter (['-interactive'])`

createJ2C Activation Spec	Use the **createJ2C Activation Spec** command to create a Java 2 Connector (J2C) activation specification under a J2C resource adapter and the attributes that you specify. Use the messageListenerType parameter to indicate the activation specification that is defined for the J2C resource adapter.	J2C Resource Adapter object ID	• Parameters: **- messageListenerType** Identifies the activation specification for the J2C activation specification to be created. Use this parameter to identify the activation specification template for the J2C resource adapter that you specify. **- name** Indicates the name of the J2C activation specification that you are creating. **- jndiName** Indicates the name of the Java Naming and Directory Interface (JNDI). **- destinationJndiName** Indicates the name of the Java Naming and Directory Interface (JNDI) of corresponding destination. **- authenticationAlias** Indicates the authentication alias of the J2C activation specification that you are creating. **- description** Description of the created J2C activation specification. • Returns: J2CActivationSpec object ID	**Batch mode example usage:** • Using Jacl: `$AdminTask createJ2CActivationSpec $ra {-name J2CActSpec -jndiName eis/ActSpec1 -messageListenerType javax.jms.MessageListener }` • Using Jython string: `AdminTask.createJ2CActivationSpec(ra, '[-name J2CActSpec -jndiName eis/ActSpec1 -messageListenerType javax.jms.MessageListener]')` • Using Jython list: `AdminTask.createJ2CActivationSpec(ra, ['-name', 'J2CActSpec', '-jndiName', 'eis/ActSpec1', '-messageListenerType', 'javax.jms.MessageListener'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createJ2CActivationSpec {-interactive}` • Using Jython string: `AdminTask.createJ2CActivationSpec ('[-interactive]')` • Using Jython list: `AdminTask.createJ2CActivationSpec (['-interactive'])`

createJ2C AdminObject	Use the **createJ2C AdminObject** command to create an administrative object under a resource adapter with attributes that you specify. Use the administrative object interface to indicate the administrative object that is defined in the resource adapter.	J2C Resource Adapter object ID	• Parameters: **-adminObjectInterface** Specifies the administrative object interface to identify the administrative object for the resource adapter that you specify. This parameter is required. **-name** Indicates the name of the administrative object. **-jndiName** Specifies the name of the Java Naming and Directory Interface (JNDI). **-description** Description of the created J2C admin object. • Returns: J2CAdminObject object ID	**Batch mode example usage:** • Using Jacl: `$AdminTask createJ2CAdmin Object $ra {-adminObject Interface fvt.adapter.mess age.FVTMessageProvider -name J2CA01 -jndiName eis/J2CA01}` • Using Jython string: `AdminTask.createJ2CAdminOb ject(ra, '[-adminObjectIn terface fvt.adapter.messag e.FVTMessageProvider -name J2CA01 -jndiName eis/J2CA01]')` • Using Jython list: `AdminTask.createJ2CAdminObj ect(ra, ['-adminObjectInte rface', 'fvt.adapter.messa ge.FVTMessageProvider', '-name', 'J2CA01', '-jndiName', 'eis/J2CA01'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createJ2CAdmin Object {-interactive}` • Using Jython string: `AdminTask.createJ2CAdmin Object ('[-interactive]')` • Using Jython list: `AdminTask.createJ2CAdmin Object (['-interactive'])`

createJ2C Connection Factory	Use the **createJ2C Connection Factory** command to create a Java 2 connection factory under a Java 2 resource adapter and the attributes that you specify. Use the connection factory interfaces to indicate the connection definitions that are defined for the Java 2 resource adapter.	J2C Resource Adapter object ID	• Parameters: **-connectionFactoryInterface** Identifies the connection definition for the Java 2 resource adapter that you specify. This parameter is required. **-name** Indicates the name of the connection factory. **-jndiName** Indicates the name of the Java Naming and Directory Interface (JNDI). **-description** Description of the created J2C connection factory. • Returns: The J2C connection factory object ID.	**Batch mode example usage:** • Using Jacl: `$AdminTask createJ2CConne ctionFactory $ra {-connec tionFactoryInterfaces javax.sql.DataSource -name J2CCF1 -jndiName eis/J2CCF1}` • Using Jython string: `AdminTask.createJ2CConne ctionFactory(ra, '[-conn ectionFactoryInterfaces javax.sql.DataSource -name J2CCF1 -jndi Name eis/J2CCF1]')` • Using Jython list: `AdminTask.createJ2CConnec tionFactory(ra, ['-connect ionFactoryInterfaces', 'javax.sql.DataSource', '-name', 'J2CCF1', '-jndiName', 'eis/J2CCF1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createJ2CConne ctionFactory {-interactive}` • Using Jython string: `AdminTask.createJ2CConnec tionFactory ('[-interactive]')` • Using Jython list: `AdminTask.createJ2CConnect ionFactory (['-interactive'])`
listAdmin ObjectInt erfaces	Use the **listAdmin ObjectIn terfaces** command to list the administrative object interfaces that are defined under the resource adapter that you specify.	J2C Resource Adapter object ID	• Parameters: None • Returns: A list of administrative object interfaces.	**Batch mode example usage:** • Using Jacl: `$AdminTask listAdminObjectInterfaces $ra` • Using Jython string: `AdminTask.listAdminObjectInterfaces(ra)` • Using Jython list: `AdminTask.listAdminObjectInterfaces(ra)` **Interactive mode example usage:** • Using Jacl: `$AdminTask listAdminObject Interfaces {-interactive}` • Using Jython string: `AdminTask.listAdminObject Interfaces ('[-interactive]')` • Using Jython list: `AdminTask.listAdminObjectIn terfaces (['-interactive'])`

listConnectionFactoryInterfaces	Use the **listConnectionFactoryInterfaces** command to list all of the connection factory interfaces that are defined under the Java 2 connector resource adapter that you specify.	J2C Resource Adapter object ID	• Parameters: None • Returns: A list of connection factory interfaces.	**Batch mode example usage:** • Using Jacl: `$AdminTask listConnectionFactoryInterfaces $ra` • Using Jython string: `AdminTask.listConnectionFactoryInterfaces(ra)` • Using Jython list: `AdminTask.listConnectionFactoryInterfaces(ra)` **Interactive mode example usage:** • Using Jacl: `$AdminTask listConnectionFactoryInterfaces {-interactive}` • Using Jython string: `AdminTask.listConnectionFactoryInterfaces ('[-interactive]')` • Using Jython list: `AdminTask.listConnectionFactoryInterfaces (['-interactive'])`
listJ2CActivationSpecs	Use the **listJ2CActivationSpecs** command to list the activation specifications that are contained under the resource adapter and message listener type that you specify.	J2C Resource Adapter object ID	• Parameters: **-messageListenerType** Specifies the message listener type for the resource adapter for which you are making a list. This parameter is required. • Returns: A list of activation specifications that has specified messageListener type.	**Batch mode example usage:** • Using Jacl: `$AdminTask listJ2CActivationSpecs $ra {-messageListenerType javax.jms.MessageListener}` • Using Jython string: `AdminTask.listJ2CActivationSpecs(ra, '[-messageListenerType javax.jms.MessageListener]')` • Using Jython list: `AdminTask.listJ2CActivationSpecs(ra, ['-messageListenerType', 'javax.jms.MessageListener'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listJ2CActivationSpecs {-interactive}` • Using Jython string: `AdminTask.listJ2CActivationSpecs ('[-interactive]')` • Using Jython list: `AdminTask.listJ2CActivationSpecs (['-interactive'])`

listJ2CAdminObjects	Use the **listJ2CAdminObjects** command to list administrative objects that contain the administrative object interface that you specify.	J2C Resource Adapter object ID	• Parameters: **-adminObjectInterface** Specifies the administrative object interface for which you want to list. This parameter is required. • Returns: A list of administrative objects that has specified adminObjectInterface.	**Batch mode example usage:** • Using Jacl: `$AdminTask listJ2CAdminObjects $ra {-adminObjectInterface fvt.adaptor.message.FVTMessageProvider}` • Using Jython string: `AdminTask.listJ2CAdminObjects(ra, '[-adminObjectInterface fvt.adaptor.message.FVTMessageProvider]')` • Using Jython list: `AdminTask.listJ2CAdminObjects(ra, ['-adminObjectInterface', 'fvt.adaptor.message.FVTMessageProvider'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listJ2CAdminObjects {-interactive}` • Using Jython string: `AdminTask.listJ2CAdminObjects ('[-interactive]')` • Using Jython list: `AdminTask.listJ2CAdminObjects (['-interactive'])`

listJ2CConnectionFactories	Use the **listJ2CConnectionFactories** command to list the Java 2 connector connection factories under the resource adapter and connection factory interface that you specify.	J2C Resource Adapter object ID	• Parameters: **-connectionFactoryInterface** Indicates the name of the connection factory that you want to list. This parameter is required. • Returns: A list of J2C connectionFactory that has the specified connectionFactoryInterface.	**Batch mode example usage:** • Using Jacl: `$AdminTask listJ2CConnectionFactories $ra {-connectionFactoryInterface javax.sql.DataSource}` • Using Jython string: `AdminTask.listJ2CConnectionFactories(ra, '[-connectionFactoryInterface javax.sql.DataSource]')` • Using Jython list: `AdminTask.listJ2CConnectionFactories(ra, ['-connectionFactoryInterface', 'javax.sql.DataSource'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listJ2CConnectionFactories {-interactive}` • Using Jython string: `AdminTask.listJ2CConnectionFactories ('[-interactive]')` • Using Jython list: `AdminTask.listJ2CConnectionFactories (['-interactive'])`
listMessageListenerTypes	Use the **listMessageListenerTypes** command to list the message listener types that are defined under the resource adapter that you specify.	J2C Resource Adapter object ID	• Parameters: None • Returns: A list of message listener types.	**Batch mode example usage:** • Using Jacl: `$AdminTask listMessageListenerTypes $ra` • Using Jython string: `AdminTask.listMessageListenerTypes(ra)` • Using Jython list: `AdminTask.listMessageListenerTypes(ra)` **Interactive mode example usage:** • Using Jacl: `$AdminTask listMessageListenerTypes {-interactive}` • Using Jython string: `AdminTask.listMessageListenerTypes ('[-interactive]')` • Using Jython list: `AdminTask.listMessageListenerTypes (['-interactive'])`

Maintaining Version 5 default messaging resources

This topic is the entry-point into a set of topics about maintaining messaging resources provided for WebSphere Application Server Version 5 applications by the default messaging provider.

WebSphere application server Version 5 applications can use JMS resources provided by the default messaging provider of WebSphere application server version 6. You can use the WebSphere administrative console to manage the JMS connection factories and destinations for WebSphere Application Server Version 5 applications. Such JMS resources are maintained as *V5 Default Messaging* resources.

V5 Default Messaging provides a JMS transport to a messaging engine of a service integration bus that supports the default messaging provider of WebSphere Application Server version 6. The messaging engine emulates the service of a Version 5 JMS server.

You can also use the administrative console to manage a JMS server on a Version 5 node.

For more information about maintaining Version 5 default messaging resources, see the following topics:
- "Listing Version 5 default messaging resources"
- "Configuring Version 5 default JMS resources" on page 875
-
- "Configuring authorization security for a Version 5 default messaging provider" on page 878

Listing Version 5 default messaging resources

Use the WebSphere administrative console to list JMS resources for the Version 5 default messaging provider, for administrative purposes.

You use the WebSphere administrative console to list JMS resources, if you want to view, modify or delete any of the following resources:
- Connection factory (unified)
- Queue connection factory
- Topic connection factory
- Queue
- Topic
- Activation specification

When you use the Administrative Console to locate these resources, two different navigation pathways are available:
- Provider-centric navigation. This lets you view all providers (or just those for a specified scope if required), then navigate to a specific resource for a specific provider. This is the traditional way of navigating to a resource when you know which provider owns it. Any navigation that starts with **Resources** → **JMS** → **JMS Providers** is provider-centric.
- Resource-centric navigation lets you view all resources of a specified type, then navigate to a resource. This is useful if you want to find a resource, but you do not know which provider owns it (you can list all resources of a given type across all scopes, for all providers, in a single panel). Any navigation that follows the pattern **Resources** → **JMS** → **[resource]**, where [resource] is one of the resource types listed above is resource-centric.

To use provider-centric navigation, for example to navigate to a specified connection factory, complete the following steps:
1. Start the WebSphere administrative console.
2. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**. This opens the providers collection which lists all currently configured providers across all scopes (you can modify the scope if required).
3. From the providers collection, select the required provider. This opens the configuration tab for that provider. The configuration tab contains a set of links to all the resources owned by that provider.
4. From the configuration tab, click the link for a resource type, for example the connection factories link. This opens the connection factories collection which lists all the connection factories for that provider.

5. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties

To use resource-centric navigation, for example to navigate to a specified connection factory, complete the following steps:

1. Start the WebSphere administrative console.
2. In the navigation pane, expand **Resources** → **JMS** → **Connection factories**. This opens the connection factories collection which lists all the connection factories across all providers.
3. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties.

You can use either of these navigation pathways to locate resources of any type.

JMS provider settings

Use this panel to view the configuration properties of a selected JMS provider. You cannot change the properties of a default messaging provider or a WebSphere MQ messaging provider.

To view this page, use the administrative console to complete one of the following steps:

- In the navigation pane, click **Resources** → **JMS** → **JMS Providers**. This displays a list of JMS providers in the content pane. For each JMS provider in the list, the entry indicates the *scope* level at which JMS resource definitions are visible to applications. You can create the same type of JMS provider at different Scope settings, to offer JMS resources at different levels of visibility to applications.
- If you want to manage JMS resources that are defined at a different scope setting, change the **Scope** setting to the required level.
- In the Providers column of the list displayed, click the name of a JMS provider.

If you want to browse or change JMS resources of the JMS provider, click the link for the type of resource under Additional Properties. For more information about the administrative console panels for the types of JMS resources, see the related topics.

For default messaging providers and WebSphere MQ messaging providers, only the scope, name, and description properties are displayed for information only. You cannot change these properties.

For another type of JMS provider that you have defined yourself, extra properties are displayed that you can change.

The default messaging provider is installed and runs as part of WebSphere Application Server, and is based on service integration technologies. For more information, see Using the default messaging provider. For information about V5 default messaging resources, see Listing Version 5 default messaging resources.

Scope:

The level to which this resource definition is visible; the cell, node, or server level.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes. For more information about the scope setting, see Scope settings.

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String

Default

- Default messaging provider.

  For JMS resources to be provided by a service integration bus, as part of WebSphere Application Server.
- *My JMSprovider*

  For JMS resources to be provided by your own JMS provider; not the default messaging provider or WebSphere MQ. You assign the name, in this example “My JMSprovider”, when you define the JMS provider to WebSphere Application Server. You must have installed and configured your own JMS provider before applications can use the JMS resources.
- WebSphere MQ messaging provider

  For JMS resources to be provided by WebSphere MQ. You must have installed and configured WebSphere MQ before applications can use the JMS resources.
- V5 default messaging provider.

  For JMS resources to be provided by a WebSphere Application Server version 5 node in a deployment manager cell.

Description:

A description of the JMS provider, for administrative purposes within WebSphere Application Server.

Data type String

Classpath:

A list of paths or JAR file names which together form the location for the JMS provider classes. Each class path entry is on a separate line (separated by using the Enter key) and must not contain path separator characters (such as ’;’ or ’: ’). Class paths can contain variable (symbolic) names to be substituted using a variable map. Check your driver installation notes for specific JAR file names that are required.

This property does not apply to default messaging providers or WebSphere MQ providers.

Data type String

Native library path:

An optional path to any native libraries (*.dll, *.so). Each native path entry is on a separate line (separated by using the Enter key) and must not contain path separator characters (such as ’;’ or ’: ’). Native paths can contain variable (symbolic) names to be substituted using a variable map.

This property does not apply to default messaging providers or WebSphere MQ providers.

Data type String

External initial context factory:

The Java classname of the initial context factory for the JMS provider.

This property does not apply to default messaging providers or WebSphere MQ providers.

For example, for an LDAP service provider the value has the form: `com.sun.jndi.ldap.LdapCtxFactory`.

Data type	String
Default	Null

External provider URL:

The JMS provider URL for external JNDI lookups.

This property does not apply to default messaging providers or WebSphere MQ providers.

For example, an LDAP URL for a messaging provider has the form: `ldap://hostname.company.com/contextName`.

Data type	String
Default	Null

Version 5 JMS server collection

On a WebSphere Application Server Version 5 node, a JMS server provides the functions of the JMS provider. Use this panel to list JMS servers on WebSphere Application Server Version 5 nodes within the administration domain, or to select a JMS server to view or change its configuration properties.

There can be at most one JMS server on each Version 5 node in the administration domain, and any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

To view this page, use the administrative console to complete the following step:
1. In the navigation pane, select **Servers** → **Version 5 JMS Servers**.

To browse or change the properties of a JMS server, select its name in the list displayed.

To act on one or more of the JMS servers listed, click the check box next to the server name, then use the buttons provided.

Version 5 JMS server settings:

The JMS functions on a Version 5 node within a deployment manager cell are served by a JMS server. Use this panel to view or change the configuration properties of the selected JMS server.

JMS servers are supported only to aid migration of WebSphere Application Server Version 5 nodes to WebSphere Application Server version 6.

You can use this panel to configure a general set of JMS server properties, which add to the default values of properties configured automatically for the Version 5 default messaging provider.

Note: JMS servers make use of WebSphere MQ properties and, in general, the default values of those properties are adequate. However, if you are running high messaging loads, you may need to change some WebSphere MQ properties; for example, properties for log file locations, file pages, and buffer pages. For more information about configuring WebSphere MQ properties, see the *WebSphere MQ System Administration* book, SC33-1873, which is available from the IBM Publications Center or from the WebSphere MQ collection kit, SK2T-0730.

Name:

The name by which the JMS server is known for administrative purposes within IBM WebSphere Application Server.

This name should not be changed.

Data type	String
Units	Not applicable
Default	WebSphere Internal JMS Server
Range	Not applicable

Description:

A description of the JMS server, for administrative purposes within IBM WebSphere Application Server.

This string should not be changed.

Data type	String
Default	WebSphere Internal JMS Server

Number of threads:

The number of concurrent threads to be used by the publish/subscribe matching engine

The number of concurrent threads should only be set to a small number.

Data type	Integer
Units	Threads
Default	1
Range	Greater than or equal to 1.

Queue Names:

The names of the queues hosted by this JMS server. Each queue name must be added on a separate line.

Each queue listed in this field must have a separate queue administrative object with the same administrative name. To make a queue available to applications, define a WebSphere queue and add its name to this field on the JMS Server panel for the host on which you want the queue to be hosted.

Data type	String
Units	Queue name
Range	Each entry in this field is a queue name of up to 45 characters, which must match exactly (including use of upper- and lowercase characters) the WebSphere queue administrative object defined for the queue.

Initial State:

The state that you want the JMS server to have when it is next restarted.

Data type	Enum
Default	Started

Range

Started
The JMS server is started automatically.

Stopped
The JMS server is not started automatically. If any deployed enterprise applications are to use JMS server functions provided by the JMS server, the system administrator must start the JMS server manually or select the Started value of this property then restart the JMS server.

To restart a JMS server on a Version 5 node, stop then restart that JMS server.

Version 5 WebSphere Queue connection factory collection

Use this panel to list JMS queue connection factories for point-to-point messaging, for use by WebSphere Application Server version 5 applications. These configuration properties control how connections are created between the JMS provider and the default messaging system that it uses.

This panel shows a list of the JMS queue connection factories with a summary of their configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere queue connection factories**. This displays a list of any existing JMS queue connection factories.

To define a new JMS queue connection factory, click **New**.

To browse or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check box next to the name of the connection factory, then use the buttons provided.

Version 5 WebSphere queue connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS queue connection factory for point-to-point messaging for use by WebSphere Application Server version 5 applications.

A WebSphere queue connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server version 5 applications.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. Under Additional Resources, click **WebSphere queue connection factories**. This displays a list of any existing JMS queue connection factories.
4. Click the name of the JMS queue connection factory that you want to work with.

A queue connection factory for the embedded WebSphere JMS provider has the following properties:

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String
Default Null

JNDI name:

The JNDI name that is used to bind the JMS connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type	Enum
Default	Null
Range	Pull-down list of Version 5 nodes in the WebSphere administrative domain.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere queue connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for container-managed authentication.

This field is deprecated in 6.0. The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. If the 'DefaultPrincipalMapping' login configuration is used, the associated property is a container-managed authentication alias. This field is used only in the absence of a loginConfiguration on the component resource reference. To define a new alias, see the related item J2EE Connector Architecture (J2C) authentication data entries.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere topic connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

The module used to map authentication aliases.

This field is deprecated in 6.0. The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. This field is used only in the absence of a loginConfiguration on the component resource reference.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum

Default Null

Range

ClientContainer
The client container maps authentication aliases.

WSLogin
The WSLogin module maps authentication aliases.

DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type Checkbox

Default Selected (enabled for XA coordination)

Range

Selected
The connection factory is enabled for XA-coordination of messages

Cleared
The connection factory is not enabled for XA coordination of messages

Recommended Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Session pool settings:

Use this page to configure session pool settings.

This administrative console page is common to a range of resource types; for example, JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Session pools**. For example: click **Resources** → **JMS Providers** → **V5 Default Messaging** → **WebSphere queue connection factories** → *connection_factory* → **Session pools**.

Connection Timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

The wait is necessary when the maximum value of connections (**Max Connections**) to a particular connection pool is reached . For example, if *Connection Timeout* is set to 300 and the maximum number of connections is reached, the Pool Manager waits for 300 seconds for an available physical connection. If a physical connection is *not* available within this time, the Pool Manager throws a ConnectionWaitTimeoutException. It usually does not make sense to retry the getConnection() method, because if a longer wait time is required, you should set the **Connection Timeout** setting to a higher value. Therefore, if this exception is caught by the application, the administrator should review the expected usage of the application and tune the connection pool and the database accordingly.

If Connection Timeout is set to 0, the Pool Manager waits as long as necessary until a connection is allocated (which happens when the number of connections falls below the value of **Max Connections**).

If Max Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Max Connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown.

For example, if the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

If Max Connections is set to 0, the Connection Timeout value is ignored.

For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

If clones are used, one data pool exists for each clone. Knowing the number of data pools is important when configuring the database maximum connections.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Data type	Integer
Default	10
Range	0 to max int

Min Connections:

Specifies the minimum number of physical connections to maintain.

Until this number is reached, the pool maintenance thread does not discard physical connections. However, no attempt is made to bring the number of connections up to this number. If you set a value for Aged Timeout, the minimum is not maintained. All connections with an expired age are discarded.

For example if the **Min Connections** value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the **Min Connections** setting.

Data type	Integer
Default	1
Range	0 to max int

Reap Time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if **Reap Time** is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the **Unused Timeout** and **Aged Timeout** settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in **Min Connections**. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused Timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections not in use exceeds the **Min Connections** setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the **Reap Time** value. For more information, see Reap Time.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged Timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting **Aged Timeout** to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the **Reap Timeout** value for optimal performance. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, are affected by the Reap Time value. For more information, see Reap Time.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge Policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**. JCA data sources can have either option. WebSphere Version 4.0 data sources always have a purge policy of **EntirePool**.

Data type	String

Default
FailingConnectionOnly

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and throws a *StaleConnectionException* during the next operation on that connection. Subsequent *getConnection* requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the *StaleConnectionException* is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next *getConnection* request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

Version 5 WebSphere topic connection factory collection

Use this panel to list JMS topic connection factories for publish/subscribe messaging, for use by WebSphere Application Server version 5 applications. These configuration properties control how connections are created between the JMS provider and the default messaging system that it uses.

This panel shows a list of JMS topic connection factories with a summary of their configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere topic connection factories**. This displays a list of any existing JMS topic connection factories.

To define a new JMS topic connection factory, click **New**.

To view or change the properties of a connection factory, select its name in the list displayed.

To act on one or more of the connection factories listed, click the check box next to the name of the connection factory, then use the buttons provided.

WebSphere topic connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS topic connection factory for publish/subscribe messaging by WebSphere Application Server version 5 applications.

A WebSphere topic connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server version 5 applications.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. Under Additional Resources, click **WebSphere topic connection factories**. This displays a list of any existing JMS topic connection factories.
4. Click the name of the JMS topic connection factory that you want to work with.

A JMS topic connection factory for use with the Version 5 default messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String
Default Null

JNDI name:

The JNDI name that is used to bind the topic connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type Enum
Default Null
Range Pull-down list of nodes in the WebSphere administrative domain.

Port:

Which of the two ports that connections use to connect to the JMS server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for non-persistent, non-transactional, non-durable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish/subscribe support. Therefore, any topic connection factory configured with **Port** set to `Direct` cannot be used with message-driven beans.

Data type Enum
Units Not applicable
Default QUEUED
Range
QUEUED
The listener port used for full-function JMS-compliant, publish/subscribe support.
DIRECT
The listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for publish/subscribe support.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere topic connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for container-managed authentication.

This field is deprecated in 6.0. The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. If the 'DefaultPrincipalMapping' login configuration is used, the associated property is a container-managed authentication alias. This field is used only in the absence of a loginConfiguration on the component resource reference. To define a new alias, see the related item J2EE Connector Architecture (J2C) authentication data entries.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

The module used to map authentication aliases.

This field is deprecated in 6.0. The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. This field is used only in the absence of a loginConfiguration on the component resource reference.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type	Enum
Default	Null

Range

ClientContainer
The client container maps authentication aliases.
WSLogin
The WSLogin module maps authentication aliases.
DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Clone Support:

Select this checkbox to enable clone support to allow the same durable subscription across topic clones.

Data type Enum
Default Cleared
Range

Selected
Clone support is enabled.
Cleared
Clone support is disabled.

If you select this property, you must also specify a value for the **Client ID** property.

Client ID:

The JMS client identifier used for connections to the queue manager.

Data type String
Range A valid JMS client ID

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type Checkbox
Default Selected (enabled for XA coordination)
Range

Selected
The connection factory is enabled for XA-coordination of messages
Cleared
The connection factory is not enabled for XA coordination of messages

Recommended	Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Version 5 WebSphere queue destination collection

Use this panel to list JMS queue for point-to-point messaging for use by WebSphere Application Server version 5 applications.

This panel shows a list of the JMS queue destinations with a summary of their configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere queue destinations**. This displays a list of any existing JMS queue destinations.

To define a new JMS queue destination, click **New**.

To view or change the properties of a queue destination, select its name in the list displayed.

To act on one or more of the queue destinations listed, click the check box next to the name of the queue, then use the buttons provided.

Version 5 WebSphere queue destination settings:

Use this panel to view or change the configuration properties of the selected JMS queue destination for point-to-point messaging by WebSphere Application Server version 5 applications.

A queue destination is used to configure a JMS queue of the default messaging provider for use by WebSphere Application Server version 5 applications. Connections to the queue are created by the associated V5 Default Messaging WebSphere queue connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere queue destinations**. This displays a list of any existing JMS queue destinations.
4. Click the name of the JMS queue destination that you want to work with.

A JMS queue for use with the internal WebSphere JMS provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

To enable applications to use this queue, you must add the queue name to the Queue Names field on the panel for the JMS server that hosts the queue.

Data type String

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String
Default Null

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.
NON PERSISTENT
Messages on the destination are not persistent.
PERSISTENT
Messages on the destination are persistent. When a persistent message is put to a queue, all of the message data is written to the messaging log (under the *embedded_messaging_install*\log directory) to make recovery of the message possible.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Default APPLICATION DEFINED

Range

APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.

QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Default 0
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type Enum
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

UNLIMITED
Messages on this queue have no expiry timeout, so those messages never expire.

SPECIFIED
The expiry timeout for messages on this queue is defined by the **Specified expiry** property.*If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type Integer
Units Milliseconds
Default 0

Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

Version 5 WebSphere topic destination collection

Use this panel to list JMS topic destinations for publish/subscribe messaging with the default messaging provider on a Version 5 node in the deployment manager cell.

This panel shows a list of JMS topic destinations with a summary of their configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere topic destinations**. This displays a list of any existing JMS topic destinations.

To define a new JMS topic connection factory, click **New**.

To browse or change the properties of a topic destination, select its name in the list displayed.

To act on one or more of the topic destinations listed, click the check box next to the name of the topic, then use the buttons provided.

Version 5 WebSphere topic destination settings:

Use this panel to browse or change the configuration properties of the selected JMS topic destination for publish/subscribe messaging by WebSphere application server version 5 applications.

A WebSphere topic destination is used to configure the properties of a JMS topic for the default messaging provider on a Version 5 node in the deployment manager cell. Connections to the topic are created by the associated topic connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. Under Additional Resources, click **WebSphere topic destinations**. This displays a list of any existing JMS topic destinations.
4. Click the name of the JMS topic destination that you want to work with.

A JMS topic destination for use with the Version 5 default messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the topic is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the topic into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic, for your IBM WebSphere Application Server administrative records.

Data type String

Topic:

The name of the topic as defined to the JMS provider.

Data type String
Default Null
Range The topic value can be dot notation and include wildcard characters.

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.
NON-PERSISTENT
Messages on the destination are not persistent.
PERSISTENT
Messages on the destination are persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Units Not applicable
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.
QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Default 0
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type Enum
Units Not applicable
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

UNLIMITED
Messages on this queue have no expiry timeout, so those messages never expire.

SPECIFIED
The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type Integer
Units Milliseconds
Default 0
Range Greater than or equal to 0
- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Configuring Version 5 default JMS resources

Use the following tasks to configure the JMS connection factories and destinations WebSphere application server Version 5 applications.

You only need to complete these tasks if you have WebSphere application server Version 5 applications that need to use JMS resources provided by the default messaging provider. Such JMS resources are maintained as *V5 Default Messaging* resources.
- Configuring a connection for Version 5 default messaging
- Configuring a Version 5 default JMS queue connection factory
- Configuring a Version 5 default JMS topic connection factory
- Configuring a Version 5 default JMS queue destination
- Configuring a Version 5 default JMS topic destination

Configuring a Version 5 queue connection factory

Use this task to browse or change the properties of a JMS queue connection factory for point-to-point messaging with the default messaging provider on a Version 5 node in the deployment manager cell. This task contains an optional step for you to create a new JMS queue connection factory.

To configure a JMS queue connection factory for use by WebSphere application server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. Select the Version 5 provider for which you want to configure a queue connection factory.
3. **Optional:** Change the **Scope** setting to the level at which the JMS queue connection factory is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.

4. In the content pane, under Additional Properties, click **Queue connection factories** This displays any existing JMS queue connection factories for the Version 5 messaging provider in the content pane.
5. To browse or change an existing JMS queue connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS queue connection factory is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the JMS queue connection factory into the name space.
   c. Click **Apply**. This defines the JMS queue connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
6. **Optional:** Change properties for the queue connection factory, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 JMS topic connection factory

Use this task to browse or change a JMS topic connection factory for publish/subscribe messaging by WebSphere application server Version 5 applications.

To configure a JMS topic connection factory for use by WebSphere application server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. Select the Version 5 provider for which you want to configure a topic connection factory.
3. **Optional:** Change the **Scope** setting to the level at which the JMS topic connection factory is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
4. In the content pane, under Additional Properties, click **Topic connection factories** This displays any existing JMS topic connection factories for the Version 5 messaging provider in the content pane.
5. To browse or change an existing JMS topic connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS topic connection factory is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the JMS topic connection factory into the name space.
   c. Click **Apply**. This defines the JMS topic connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
6. **Optional:** Change properties for the topic connection factory, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 WebSphere queue destination

Use this task to browse or change the properties of a JMS queue destination for point-to-point messaging by WebSphere Application Server version 5 applications. This task contains an optional step for you to create a new topic destination.

To configure a JMS queue destination for use by WebSphere application server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. **Optional:** Change the **Scope** setting to the level at which the JMS destination is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. Select the Version 5 provider for which you want to configure a queue destination.
4. In the content pane, under Additional Properties, click **Queues** This displays any existing queue destinations for the Version 5 default messaging provider in the content pane.
5. To browse or change an existing JMS queue destination, click its name in the list. Otherwise, to create a new queue destination, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this queue destination is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the queue destination into the name space.
   c. Click **Apply**. This defines the queue destination to WebSphere Application Server, and enables you to browse or change additional properties.
6. **Optional:** Change properties for the queue destination, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To make a queue destination available to applications, you need to host the queue on a JMS server. To add a new queue to a JMS server or to change an existing queue on a JMS server, you define the administrative name of the queue to the JMS server, as described in Managing Version 5 JMS servers in a deployment manager cell.
10. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 WebSphere topic destination

Use this task to browse or change the properties of a JMS topic destination for publish/subscribe messaging by WebSphere application server Version 5 applications.. This task contains an optional step for you to create a new topic destination.

To configure a JMS topic destination for use WebSphere application server version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. Select the Version 5 provider for which you want to configure a topic destination.
3. **Optional:** Change the **Scope** setting to the level at which the JMS destination is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
4. In the content pane, under Additional Properties, click **Topics** This displays any existing JMS topic destinations for the Version 5 default messaging provider in the content pane.

5. To browse or change an existing JMS topic destination, click its name in the list. Otherwise, to create a new topic destination, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this topic destination is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the topic destination into the name space.

      Topic The name of the topic in the default messaging provider, to which messages are sent.
   c. Click **Apply**. This defines the topic destination to WebSphere Application Server, and enables you to browse or change additional properties.
6. **Optional:** Change properties for the topic destination, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring authorization security for a Version 5 default messaging provider

Use this task to configure authorization security for the default messaging provider on a WebSphere Application Server version 5 node in a deployment manager cell.

To configure authorization security for the version 5 default messaging provider complete the following steps.

Note: Security for the version 5 default messaging provider is enabled when you enable global security for WebSphere Application Server on the version 5 node. For more information about enabling global security, see Enabling security for all application servers.

1. Configure authorization settings to access JMS resources owned by the embedded WebSphere JMS provider.

   Authorization to access JMS resources owned by the embedded messaging subsystem is controlled by settings in the *app_server_root*\config\cells*your_cell_name*\integral-jms-authorizations.xml file. The settings grant or deny authenticated users access to messaging resources (queues or topics). As supplied, the integral-jms-authorisations.xml file grants the following permissions:
   - Read and write permissions to all queues.
   - Pub, sub, and persist to all topics.

   To configure authorization settings, edit the integral-jms-authorisations.xml file according to the information in this topic and in that file. Please note the file is in Unicode, which requires a binary FTP to the host from a workstation.
2. Edit the queue-admin-userids element to create a list of userids with administrative access to all queues. Administrative access is needed to create queues and perform other administrative activities on queues. For example, consider the following queue-admin-userids section:

   ```
   <queue-admin-userids>
       <userid>adminid1</userid>
       <userid>adminid2</userid>
   </queue-admin-userids>
   ```

   In this example the userids adminid1 and adminid2 are defined to have administrative access to all queues.

3. Edit the queue-default-permissions element to define the default queue access permissions. These permissions are used for queues for which you do not define specific permissions (in queue sections). If this section is not specified, then access permissions exist only for those queues for which you have specifically created queue elements.

   For example, consider the following queue-default-permissions element:

   ```
   <queue-default-permissions>
     <permission>write</permission>
   </queue-default-permissions>
   ```

   In this example the default access permission for all queues is **write**. This can be overridden for a specific queue by creating a queue element that sets its access permission to **read**.

4. If you want to define specific access permissions for a queue, create a queue element, then define the following elements:

   For example, consider the following queue element:

   ```
   <queue>
     <name>q1</name>
     <public>
     </public>
     <authorize>
       <userid>useridr</userid>
       <permission>read</permission>
     </authorize>
     <authorize>
       <userid>useridw</userid>
       <permission>write</permission>
     </authorize>
     <authorize>
       <userid>useridrw</userid>
       <permission>read</permission>
       <permission>write</permission>
     </authorize>
   </queue>
   ```

   In this example for the queue q1, the userid useridr has read permission, the userid useridw has write permission, the userid useridrw has both read and write permissions, and all other userids have no access permissions (<public></public>).

5. Edit topic elements to define the access permissions for publish/subscribe topic destinations.

   For topics, you can grant and deny access permissions. Full permission inheritance is supported on topics. If you do not define specific access permissions for a userid on a specific topic then permissions are inherited first from the public permissions on that topic then from the parent topic. The inheritance of access permissions continues until the root topic from which the root permissions are assumed.

   a. If you want to define default access permissions for the root topic, edit a topic element with an empty name element. If you omit such a topic section, topics have no default topic permissions other than those defined by specific topic elements. For example, consider the following topic element for the root topic:

      ```
      <topic>
         <name></name>
         <public>
           <permission>+pub</permission>
         </public>
       </topic>
      ```

      In this example, the default access permission for all topics is set to publish. This can be overridden by other topic elements for specific topic names.

   b. If you want to define access permissions for a specific topic, create a topic element with the name for the topic then define the access permissions in the public and authorize elements of the topic element. For example, consider the following topic section:

```
<topic>
  <name>a/b/c</name>
  <public>
    <permission>+sub</permission>
  </public>
  <authorize>
    <userid>useridpub</userid>
    <permission>+pub</permission>
  </authorize>
</topic>
```

In this example, the subscribe permission is granted to anyone accessing any topic whose name starts with `a/b/c`. Also, the userid useridpub is granted publish permission for any topic whose name starts with `a/b/c`.

6. Save the integral-jms-authorizations.xml file.

If the dynamic update setting is selected, changes to the integral-jms-authorizations.xml file become active when the changed file is saved, so there is no need to stop and restarted the JMS server. If the dynamic update setting is not selected, you need to stop and restart the JMS server to make changes active.

Authorization settings for Version 5 default JMS resources

Use the integral-jms-authorisations.xml file to view or change the authorization settings for Java Message Service (JMS) resources owned by the default messaging provider on WebSphere Application Server version 5 nodes.

Authorization to access default JMS resources owned by the default messaging provider on WebSphere Application Server nodes is controlled by the following settings in the *was_install*`\config\cells\`*your_cell_name*`\integral-jms-authorisations.xml` file.

This structure of the settings in integral-jms-authorisations.xml is shown in the following example. Descriptions of these settings are provided after the example. To configure authorization settings, follow the instructions provided in Configuring authorization security for the Version 5 JMS providers

```
<integral-jms-authorizations>

  <dynamic-update>true</dynamic-update>

  <queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
  </queue-admin-userids>

  <queue-default-permissions>
    <permission>write</permission>
  </queue-default-permissions>

  <queue>
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
  </queue>

  <queue>
    <name>q2</name>
    <public>
      <permission>write</permission>
```

```
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
  </queue>

  <topic>
    <name></name>
    <public>
      <permission>+pub</permission>
    </public>
  </topic>

  <topic>
    <name>a/b/c</name>
    <public>
      <permission>+sub</permission>
    </public>
    <authorize>
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>

</integral-jms-authorizations>
```

dynamic-update: Controls whether or not the JMS Server checks dynamically for updates to this file.
true (Default) Enables dynamic update support.
false Disables dynamic update checking and improves authorization performance.

queue-admin-userids: This element lists those userids with administrative access to all Version 5 default queue destinations. Administrative access is needed to create queues and perform other administrative activities on queues. You define each userid within a separate userid sub element:
<userid>*adminid***</userid>**

Where *adminid* is a user ID that can be authenticated by IBM WebSphere Application Server.

queue-default-permissions: This element defines the default queue access permissions that are assumed if no permissions are specified for a specific queue name. These permissions are used for queues for which you do not define specific permissions (in queue elements). If this element is not specified, then no access permissions exist unless explicitly authorized for individual queues.

You define the default permission within a separate permission sub element:
<permission>*read-write***</permission>**

Where *read-write* is one of the following keywords:
read By default, userids have read access to Version 5 default queue destinations.
write By default, userids have write access to Version 5 default queue destinations.

queue: This element contains the following authorization settings for a single queue destination:
name The name of the queue.
public The default public access permissions for the queue. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the queue.

You define each default permission within a separate permission element.
authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:
userid The userid that you want to assign a specific access permission.
permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain the keyword read or write to define the access permission.

For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

topic: This element contains the following authorization settings for a single topic destination:

Each topic element has the following sub elements:

name The name of the topic, without wildcards or other substitution characters.

public The default public access permissions for the topic. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the topic.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain one of the following keywords to define the access permission:

+pub Grant publish permission

+sub Grant subscribe permission

+persist

Grant persist permission

-pub Deny publish permission

-sub Deny subscribe permission

-persist

Deny persist permission

JMS components on Version 5 nodes

To provide messaging support on a WebSphere Application Server Version 5 node, there is at most one JMS server and some number of JMS resources configured for the default messaging JMS provider on that node.

A *JMS server* on a Version 5 node serves the JMS resources (connection factories and destinations) for that node. The JMS server is managed as a separate process to application servers on the same node. Any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

A *connection factory* encapsulates the configuration properties used to create connections with the JMS provider, to enable applications to access JMS destinations.

The main components of JMS support on a Version 5 node are shown in the figure The main components of WebSphere JMS support.

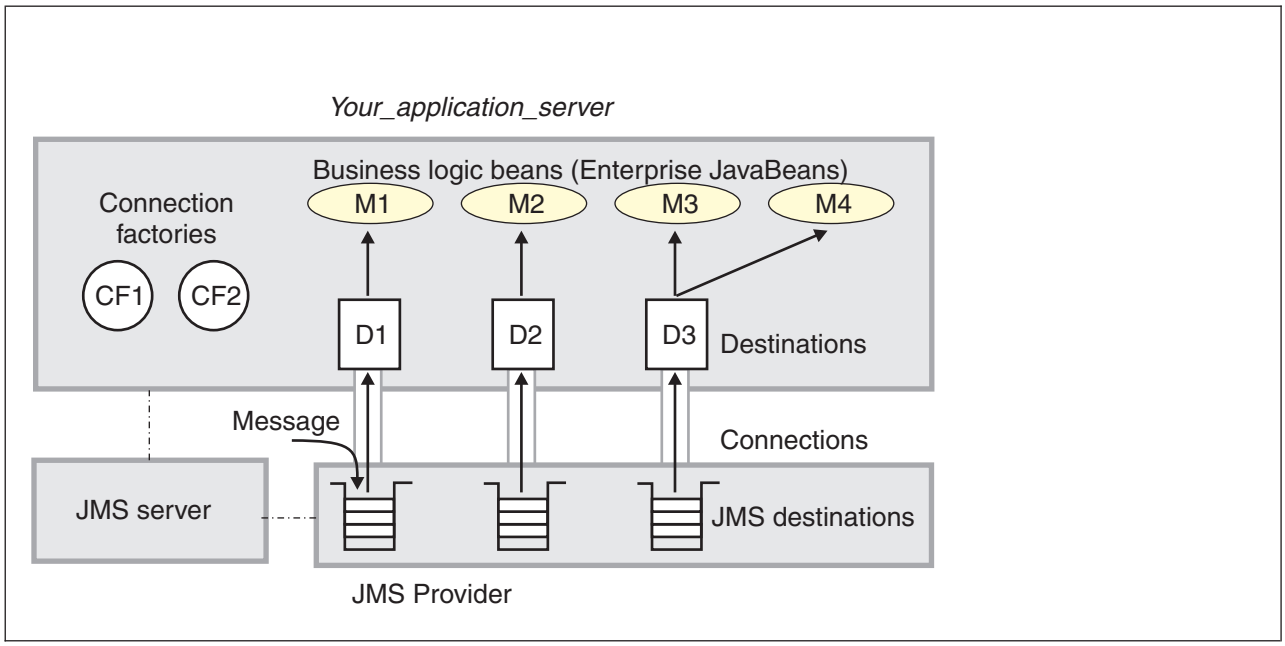


Figure 7. The main JMS components on a version 5 node. This figure shows the main JMS components on a version 5 node, from JMS provider through a connection to a destination, then to a WebSphere enterprise application (acting as a JMS client) that processes the message retrieved from the destination. For more information, see the text that accompanies this figure.

Using the JMS resources provided by WebSphere MQ

This topic is the entry-point into a set of topics about enabling WebSphere applications to use the JMS resources provided by WebSphere MQ.

WebSphere MQ can act as a JMS provider for WebSphere Application Server. WebSphere applications use the JMS 1.1 or JMS 1.0.2 interfaces to access the JMS resources provided by WebSphere MQ, and to access JMS resources provided by the default messaging provider (or a generic messaging provider). You use the WebSphere administrative console to administer the resources provided by WebSphere MQ.

In a mixed-version WebSphere Application Server cell, you can administer WebSphere MQ resources on nodes of all versions. However, some properties are not available on all versions. In this situation, the administrative console displays only the properties of that particular node.

On WebSphere Application Server Version 6.1, it is possible to connect to a WebSphere MQ network for messaging interaction using a WebSphere MQ Link, a WebSphere MQ server, or by using WebSphere MQ as an external JMS messaging provider.

For more information about using WebSphere MQ as a messaging provider for WebSphere Application Server, see the following topics:

- “Installing and configuring WebSphere MQ as a JMS provider” on page 884
- “Listing JMS resources for WebSphere MQ” on page 886
- “Configuring JMS resources for the WebSphere MQ messaging provider” on page 956

Installing and configuring WebSphere MQ as a JMS provider

If you have a WebSphere MQ messaging infrastructure, you can install and configure it as a JMS provider to WebSphere Application Server.

If you have a messaging infrastructure based on WebSphere MQ, you can connect directly using the included support for WebSphere MQ as a JMS provider.

(UNIX platforms only): Before you install WebSphere MQ on a UNIX platform, create and mount a journalized file system called /var/mqm for your messaging working data. Use a partition strategy with a separate volume for the WebSphere MQ data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm. You can also create separate file systems for your log data (var/mqm/log) and error files (var/mqm/errors). You should store log files on a different physical volume from the messaging queues (var/mqm). This ensures data integrity in the case of a hardware failure. If you are creating separate file systems, allow a minimum of 30 MB of storage for /var/mqm, 20 MB of storage for /var/mqm/log, and 4 MB of storage for /var/mqm/errors.

The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.

It is not recommended to install Rational Application Developer and WebSphere Application Server on the same machine when using WebSphere MQ.

For more information about creating file systems for WebSphere MQ, and WebSphere MQ space requirements for /var file systems, see the section “Preparing for Installation: Creating WebSphere MQ file systems” in the appropriate WebSphere MQ *Quick Beginnings* book.

For other installation prerequisites, see the following WebSphere MQ publications:

- *WebSphere MQ for Windows, Quick Beginnings*, GC34-6073
- *WebSphere MQ for AIX, Quick Beginnings*, GC34-6076
- *WebSphere MQ for Solaris, Quick Beginnings*, GC34-6075
- *WebSphere MQ for HP-UX, Quick Beginnings*, GC34-6077
- *WebSphere MQ for Linux for Intel and Linux for zSeries, Quick Beginnings*, GC34-6078

You can get these books from the WebSphere MQ messaging platform-specific books Web page at http://www-306.ibm.com/software/integration/wmq/library/

To install and configure WebSphere MQ for use as a JMS provider to IBM WebSphere Application Server, complete the following steps:

1. Install a supported version of WebSphere MQ, with the required MQ features, as described in the installation instructions provided with WebSphere MQ.

   To identify the supported version of WebSphere MQ, see the Detailed system requirements page.

   For information about installing WebSphere MQ, or migrating to a supported version of WebSphere MQ from an earlier version, see the appropriate WebSphere MQ *Quick Beginnings* book, as listed above.

   (RHEL 3.0 and SLES 8) (RHEL 3.0 and SLES 8) Before installing WebSphere MQ 5.3 on an Intel or zSeries Linux system using the 2.6 kernel, you must first download FixPack 10 or later (plus any applicable interim fixes) and follow the software prerequisite instructions at http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27006464. Otherwise, you might see a segmentation fault error when attempting to run the `mqlicense.sh -accept` command to accept the WebSphere MQ license agreement. Also, export the following variable: `LD_ASSUME_KERNEL=2.4.19`

- Red Hat Enterprise Linux (RHEL) 3.0 on Intel or s390
- SUSE Linux Enterprise Server (SLES) 8 on Intel

To see the WebSphere MQ V5.3 for Linux for Intel fix pack readme see http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27006464

To see the WebSphere MQ V5.3 for Linux for zSeries fix pack readme see http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27006464

If request metrics is enabled when using WebSphere MQ V5.3, an exception is issued and request metrics functions fail. If you use WebSphere MQ V5.3 plus CSD08, no exception is issued; however, request metrics still fails to record the Java Message Service (JMS) type request information. The solution is to apply the interim fix for WebSphere MQ V5.3 plus CSD08: Put the JAR files in the /opt/IBM/WebSphere/AppServer/lib/WMQ/java/lib directory and in the external websphere_mq_install_root/java/lib directories. For more information about this issue, see the Technote 1192026; for example, at http://www-1.ibm.com/support/docview.wss?rs=0&q1;=1192026 &uid=swg21192026.

2. If you want to use WebSphere MQ - Publish/Subscribe support, you need to provide a Publish/Subscribe broker.

   For example, you can do this by using either WebSphere MQ Event Broker or WebSphere MQ Integrator (formerly MQSeries Integrator). For more information about these products, see the following Web sites:
   - WebSphere MQ Event Broker Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#eventb
   - WebSphere MQ Integrator Web site at http://www-4.ibm.com/software/ts/mqseries/platforms/#integrator
3. Follow the WebSphere MQ instructions for verifying your installation setup.
4. AIX For AIX, see the WebSphere MQ readme.txt for additional steps.
5. Set the MQ_INSTALL_ROOT environment variable to the directory where WebSphere MQ is installed. IBM WebSphere Application Server uses the MQ_INSTALL_ROOT to locate the WebSphere MQ libraries for the WebSphere MQ JMS Provider.

This task has installed WebSphere MQ for use as a JMS provider for WebSphere Application Server.

You can configure JMS resources to be provided by WebSphere MQ, by using the WebSphere administrative console to define WebSphere MQ resources.

(UNIX platforms only) Restrict access to the messaging errors directories and logging files, by using the following commands. This is part of the procedure to secure the directories and log files needed for WebSphere MQ, as described in Securing messaging directories and log files.

1. For the /var/mqm/errors directory:

```
chmod 3777 /var/mqm/errors
chown mqm:mqm /var/mqm/errors

touch /var/mqm/errors/AMQERR01.LOG
chown mqm:mqm /var/mqm/errors/AMQERR01.LOG
chmod 666 /var/mqm/errors/AMQERR01.LOG

touch /var/mqm/errors/AMQERR02.LOG
chown mqm:mqm /var/mqm/errors/AMQERR02.LOG
chmod 666 /var/mqm/errors/AMQERR02.LOG

touch /var/mqm/errors/AMQERR03.LOG
chown mqm:mqm /var/mqm/errors/AMQERR03.LOG
chmod 666 /var/mqm/errors/AMQERR03.LOG
```

2. For the /var/mqm/qmgrs/@SYSTEM/errors directory:

```
chmod 3777 /var/mqm/qmgrs/@SYSTEM/errors
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors
```

```
touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG

touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
```

For information about how WebSphere Application Server can interact with a WebSphere MQ network, using either a WebSphere MQ Link or a WebSphere MQ server, see the following topics:

- ../concepts/cjfp0000_.dita
- ../tasks/tjfp0015_.dita

Listing JMS resources for WebSphere MQ

Use the WebSphere administrative console to list JMS resources for the WebSphere MQ provider, for administrative purposes.

You use the WebSphere administrative console to list JMS resources, if you want to view, modify or delete any of the following resources:

- Connection factory (unified)
- Queue connection factory
- Topic connection factory
- Queue
- Topic
- Activation specification

When you use the Administrative Console to locate these resources, two different navigation pathways are available:

- Provider-centric navigation. This lets you view all providers (or just those for a specified scope if required), then navigate to a specific resource for a specific provider. This is the traditional way of navigating to a resource when you know which provider owns it. Any navigation that starts with **Resources** → **JMS** → **JMS Providers** is provider-centric.
- Resource-centric navigation lets you view all resources of a specified type, then navigate to a resource. This is useful if you want to find a resource, but you do not know which provider owns it (you can list all resources of a given type across all scopes, for all providers, in a single panel). Any navigation that follows the pattern **Resources** → **JMS** → **[resource]**, where [resource] is one of the resource types listed above is resource-centric.

To use provider-centric navigation, for example to navigate to a specified connection factory, complete the following steps:

1. Start the WebSphere administrative console.
2. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**. This opens the providers collection which lists all currently configured providers across all scopes (you can modify the scope if required).
3. From the providers collection, select the required provider. This opens the configuration tab for that provider. The configuration tab contains a set of links to all the resources owned by that provider.
4. From the configuration tab, click the link for a resource type, for example the connection factories link. This opens the connection factories collection which lists all the connection factories for that provider.
5. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties

To use resource-centric navigation, for example to navigate to a specified connection factory, complete the following steps:

1. Start the WebSphere administrative console.
2. In the navigation pane, expand **Resources** → **JMS** → **Connection factories**. This opens the connection factories collection which lists all the connection factories across all providers.
3. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties.

You can use either of these navigation pathways to locate resources of any type.

WebSphere MQ connection factory collection

The unified JMS connection factories configured in the WebSphere MQ messaging provider, for both point-to-point and publish/subscribe messaging.

This panel shows a list of the WebSphere MQ unified JMS connection factories with a summary of their configuration properties. In a deployment manager cell, these connection factories are not available for Version 5 nodes.

This type of connection factory is sometimes called a "unified" or "domain-independent" JMS connection factory, and supports the JMS 1.1 domain-independent interfaces (referred to as the ″common interfaces″ in the JMS specification). This enables applications to use the same, common, interfaces for both point-to-point and publish/subscribe messaging. A unified JMS connection factory also supports the domain-specific (queue and topic) interfaces, as used in JMS 1.0.2, so applications can still use those interfaces.

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

1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. In the content pane, ensure that the scope is set to cell scope or to node or server scope for a Version 6 node.
3. In the content pane, under Additional Resources, click **WebSphere MQ connection factories**. This displays a list of any existing JMS queue connection factories.

To define a new connection factory, click **New**.

To view or change the properties of a queue connection factory, click its name in the list displayed.

To act on one or more of the connection factories listed, select the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ connection factory settings:

Use this panel to view or change the configuration properties of the selected JMS connection factory for use with WebSphere MQ as a JMS provider. These configuration properties control how connections are created to associated JMS queues and topics.

This type of connection factory is sometimes called a "unified" or "domain-independent" JMS connection factory, and supports the JMS 1.1 domain-independent interfaces (referred to as the ″common interfaces″ in the JMS specification). This enables applications to use the same, common, interfaces for both point-to-point and publish/subscribe messaging. A unified JMS connection factory also supports the domain-specific (queue and topic) interfaces, as used in JMS 1.0.2, so applications can still use those interfaces.

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

1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. In the content pane, ensure that the scope is set to cell scope, or to node or server scope for a Version 6 node.

3. In the content pane, under Additional Resources, click **WebSphere MQ connection factories**.
4. Click the name of the JMS connection factory that you want to work with.

A unified JMS connection factory for the WebSphere MQ JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book. and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- For more information about setting SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

JNDI name:

The JNDI name that is used to bind the connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Authentication mechanism preference:

The authentication mechanism to be used for connections to WebSphere MQ created by this connection factory.

If WebSphere MQ is not configured to support the authentication mechanism preference, it is ignored.

Data type Enum
Default BASIC PASSWORD
Range

BASIC PASSWORD
Authentication is performed based on a user ID and password provided by an authentication alias. The authentication alias used is obtained from one of the following properties:

- Component-managed Authentication Alias, for application-managed authentication.
- Container-managed Authentication Alias, for container-managed authentication.

KerbV5
Authentication is performed based on SSL certificates.

Component-managed authentication alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with WebSphere MQ. For example, the default Windows user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere MQ queue connection factory must specify a user ID no longer than 12 characters.

2. If you want to use Bindings transport mode on JMS queue connections to WebSphere MQ, you set the **Transport type** property to BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Container-managed authentication alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with WebSphere MQ. For example, the default Windows user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere MQ queue connection factory must specify a user ID no longer than 12 characters.
2. If you want to use Bindings transport mode on JMS queue connections to WebSphere MQ, you set the **Transport type** property to BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Mapping-configuration alias:

The module used to map authentication aliases.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type	Enum
Default	Null

Range

ClientContainer
The client container maps authentication aliases.

WSLogin
The WSLogin module maps authentication aliases.

DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Manage cached handles:

Whether or not cached handles (held in inst vars in a bean) should be tracked by the container.

Tracking handles can cause a large performance overhead if used at runtime; however, for debugging purposes it can be useful to enable handle management.

Data type Check box
Default Cleared
Range

Cleared
Cached handles are not tracked by the container.

Selected
Cached handles are tracked by the container. You should select this option only for debugging purposes, because this can cause a large performance overhead.

Log missing transaction contexts:

Whether or not the container logs when there is a missing transaction context at the time that a connection is created.

Data type Check box
Default Selected
Range

Selected
When a connection is created, any missing transaction contexts are logged in the activity log.

Cleared
Missing transaction contexts are not logged.

Queue manager:

The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type String
Default Null
Range A valid WebSphere MQ queue manager name, as 1 through 48 ASCII characters

Host:

The name of the host on which the WebSphere MQ queue manager runs, for client connection only.

Data type String
Default Null
Range A valid TCP/IP hostname

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type Integer
Default Null
Range A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type String
Default Null
Range 1 through 20 ASCII characters

Transport type:

Specifies whether to use the WebSphere MQ client connection or JNI bindings for connection to the WebSphere MQ queue manager. WebSphere MQ as the JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF nonpersistent, non-durable, non-transactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type Enum
Units Not applicable
Default BINDINGS
Range
BINDINGS
JNI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and comes at some security risks that should be addressed through the use of EJB roles.
CLIENT
WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.
Recommended BINDINGS is faster by 30% or more, but it lacks security. When you have security concerns, CLIENT is more desirable than BINDINGS.

Model queue definition:

The name of the model queue definition that can be used by the queue manager to create temporary queues if a queue requested does not already exist.

Data type String
Default Null
Range 1 through 48 ASCII characters

Client ID:

The JMS client identifier used for connections to the WebSphere MQ queue manager.

Data type String
Default Null

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

The term 'null' means leave blank; if you do this, a null value is passed and the default WebSphere MQ CCSID value is used.

Data type String
Units Integer
Default Null
Range 1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at the IBM Publications Center, http://www-306.ibm.com/software/integration/wmq/library/, or from the WebSphere MQ collection kit, SK2T-0730.

Enable message retention:

Whether or not unwanted messages are left on the queue. If this option is not enabled, unwanted messages are dealt with according to their disposition options.

Data type Check box
Default Cleared
Range
Selected
Unwanted messages are left on the queue.
Cleared
Unwanted messages are dealt with according to their disposition options.

XA enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are not used in the same transaction.

If you clear this property, the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

Data type Check box
Units Not applicable
Default Selected (XA enabled)
Range
Selected
The connection factory is for XA-coordination of messages
Cleared
The connection factory is for non-XA coordination of messages
Recommended Do not enable XA when the message queue received is the only resource in the transaction. Enable XA when other resources, including other queues or topics, are involved.

Enable return methods during shutdown:

Whether or not applications return from a method call if the queue manager has entered a controlled shutdown.

Data type Checkbox
Default Selected
Range
Selected
Applications return from a method call if the queue manager has entered a controlled shutdown.
Cleared
Applications do not return from a method call if the queue manager has entered a controlled shutdown.

Local server address:

The range of local ports to be used when making a connection to a WebSphere MQ queue manager

If a JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can use this property to specify a port, or a range of points, that the application can bind to.

Data type String
Default Null

Range

A string in the format:

```
[ip-addr][(low-port[,high-port])]
```

For example:

- 9.20.4.98

  The channel binds to address 9.20.4.98 locally
- 9.20.4.98(1000)

  The channel binds to address 9.20.4.98 locally and uses port 1000
- 9.20.4.98(1000,2000)

  The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000
- (1000)

  The channel binds to port 1000 locally
- (1000,2000)

  The channel binds to a port in the range 1000 to 2000 locally

You can specify a host name instead of an IP address.

For direct connections, this property applies only when multicast is used and the value of the property must not contain a port number. If it does contain a port number, the connection is rejected. Therefore, the only valid values of the property are null, an IP address, or a host name.

Polling interval:

The interval, in milliseconds, between scans of all receivers during asynchronous message delivery

Data type	Integer
Units	milliseconds
Default	5000
Range	1 through 2147483647

Rescan interval:

The interval in milliseconds between which a topic is scanned to look for messages that have been added to a topic out of order.

This interval controls the scanning for messages that have been added to a topic out of order with respect to a WebSphere MQ browse cursor.

Data type	Integer
Units	milliseconds
Default	5000
Range	1 through 2147483647

SSL cipher suite:

The cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the **Channel** property.

You must set this property if the **SSL Peer Name** property is to be set.

SSL CRL:

A list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. (Use of this property requires a WebSphere MQ JVM at Java 2 version 1.4.)

The value is a space-delimited list of entries of the form:
`ldap://hostname:[port]`

optionally followed by a single / (forward slash). If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section "Working with Certificate Revocation Lists" in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL peer name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connect-time.

The SSL Peer Name property is ignored if **SSL cipher suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:
`CN=QMGR.*, OU=IBM, OU=WEBSPHERE`

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the *WebSphere MQ Security* book; for example, the section "Distinguished Names" at http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas010p.htm#HDRDCDN.

Temporary queue prefix:

The prefix that is used for names of temporary JMS queues created by applications that use this connection factory.

Data type	String
Default	Null

Enable MQ connection pooling:

Whether or not to use WebSphere MQ connection pooling.

Data type	Checkbox
Default	Selected

Range

Selected
The connection factory uses WebSphere MQ connection pooling. When a connection is no longer required, instead of destroying it, it can be pooled, and later reused. This can provide a substantial performance enhancement for repeated connections to the same queue manager.

Cleared
The connection factory does not use WebSphere MQ connection pooling. When a connection is no longer required, it is destroyed. To use the same queue manager a new connection is created.

Broker control queue:

The name of the publish/subscribe broker's control queue, to which publisher and subscriber applications send all command messages (except publications and requests to delete publications).

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Broker queue manager:

The name of the WebSphere MQ queue manager that provides the publish/subscribe message broker.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Broker publication queue:

The name of the broker's input queue (stream queue) that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker subscription queue:

The name of the broker's queue from which non-durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Broker CC subscription queue:

The name of the broker's queue from which non-durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Broker version:

Whether the message broker is provided by the WebSphere MQ MA0C Supportpac or newer versions of WebSphere message broker products.

Data type	Enum
Default	Advanced
Range	**Advanced** The message broker is provided by newer versions of WebSphere message broker products, such as WebsSphere MQ Integrator and Event Broker. **Basic** The message broker is provided by the WebSphere MQ MA0C SupportPac (MQSeries - Publish/Subscribe) or MQSI working in MA0C compatibility mode.

Publish/subscribe cleanup level:

The level of cleanup provided by the Publish/subscribe cleanup utility

To avoid the problems associated with non-graceful closure of subscriber objects, WebSphere MQ as a JMS provider provides a Publish/Subscribe cleanup utility that attempts to detect any earlier JMS publish/subscribe problems. If a large number of problems are detected, some performance degradation may be observed while resources are cleaned up. This utility runs transparently on a background thread and should not affect other WebSphere MQ operations.

Data type	Enum
Default	SAFE

Range

SAFE The Cleanup thread attempts to remove unconsumed subscription messages, or temporary queues, for failed subscriptions. This mode of cleanup does not interfere with the operation of other JMS applications.

ASPROP
The style of cleanup to use is determined by the system property com.ibm.mq.jms.cleanup, which is queried at JVM startup. This property can be set on the java command-line using the -D option, and should be set to NONE, SAFE or STRONG. Any other value causes an exception. If not set, the property defaults to SAFE. This allows easy JVM-wide change to the Cleanup level without needing to update every topic connection factory used by the system.

NONE In this special mode, no cleanup is performed; and no cleanup thread exists. Additionally, if the application is using the single-queue approach, unconsumed messages can be left on the queue.

This option can be useful if the application is distant from the queue manager, and especially if it only publishes rather than subscribes. However, some application should perform cleanup on the queue manager to deal with any unconsumed messages - this could be a JMS application with CLEANUP(SAFE) or CLEANUP(STRONG), or the WebSphere MQ manual cleanup utility.

STRONG
The cleanup thread performs as CLEANUP(SAFE), but also clears the SYSTEM.JMS.REPORT.QUEUE of any unrecognized messages.

Publish/subscribe cleanup interval:

The interval, in milliseconds, between background executions of the publish/subscribe cleanup utility.

Data type Integer
Default 60000
Range 1 through 2147483647

Message selection:

Whether message selection is done at the broker or client.

Data type Enum
Default BROKER
Range

BROKER
Message selection is done at the broker.

CLIENT
Message selection is done at the client.

Publish acknowledgement interval:

The interval, in number of messages, between publish requests that require acknowledgement from the broker.

Data type Integer
Default 25
Range 1 through 2147483647

Enable sparse subscriptions:

Select this option to support subscriptions that receive infrequent matching messages.

Data type Checkbox
Default Cleared
Range

Selected
Subscriptions can receive infrequent matching messages. This value requires that the subscription queue can be opened for browse.

Cleared
Sparse subscriptions are not supported. Subscriptions receive frequent matching messages.

Publish/subscribe status interval:

The interval, in milliseconds, between transactions to refresh publish/subscribe status.

Data type Integer
Default 60000
Range 1 through 2147483647

Persistent subscriptions store:

Where WebSphere MQ stores persistent data relating to active JMS subscriptions.

Data type Enum
Default MIGRATE

Range

MIGRATE
This option dynamically selects the queue-based or broker-based subscription store based on the levels of queue manager and publish/subscribe broker installed. If both queue manager and broker are capable of supporting SUBSTORE(BROKER), this behaves as SUBSTORE(BROKER); otherwise it behaves as SUBSTORE(QUEUE). Additionally, SUBSTORE(MIGRATE) transfers durable subscription information from the queue-based subscription store to the broker-based store.

QUEUE
Subscription information is stored on SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE on the local queue manager.

BROKER
Subscription information is stored by the publish/subscribe broker used by the application. This option requires recent levels of queue manager and publish/subscribe broker. This subscription store requires recent levels of both queue manager and publish/subscribe broker. It is designed to provide improved resilience.

Enable multicast transport:

Whether or not this connection factory uses multicast transport.

With multicast, messages are delivered to all consumers. This is useful in environments where there are a large number of clients that all want to receive the same messages, because with multicast only one copy of each message is sent. Multicast reduces the total amount of network traffic. Reliable multicast is standard multicast with a reliability layer added.

Data type
Enum

Default
NOTUSED

Range

NOTUSED
This connection factory does not use multicast transport.

ENABLED
This connection factory uses multicast transport, but does not provide a reliable multicast connection.

ENABLED_IF_AVAILABLE
This connection factory uses multicast transport if the message broker supports it.

ENABLED_RELIABLE
This connection factory uses reliable multicast transport

ENABLED_RELIABLE_IF_AVAILABLE
This connection factory uses reliable multicast transport if the message broker supports it.

Enable clone support:

Select this check box to enable clone support to allow the same durable subscription across topic clones.

Data type check box
Default Cleared
Range
Selected
Clone support is enabled.
Cleared
Clone support is disabled.

If you select this property, you must also specify a value for the **Client ID** property.

Direct broker authentication:

Whether the broker uses basic or certificate-based authentication for direct connections.

This property selects the authentication on a direct connections (if the TRANSPORT property is set to DIRECT).

Data type Enum
Default NONE
Range

NONE Direct broker authentication is not used.

PASSWORD
Password-based authentication is used for direct connections. Authentication is performed based on a user ID and password provided by an authentication alias. The authentication alias used is obtained from one of the following properties:

- Component-managed Authentication Alias, for application-managed authentication.
- Container-managed Authentication Alias, for container-managed authentication.

CERTIFICATE
Certificate-based authentication is used for direct connections. The SSLPEERNAME and SSLCRL properties are used to perform the authentication checks.

You can use certificate-based authentication when connecting directly to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker.

Proxy host name:

Host name of the Web Scale proxy host.

A direct connection is made to the proxy server, which forwards the connection request to the message broker.

If the TRANSPORT property is set to DIRECT, the type of connection to the message broker depends on the value of this property, according to the following rules:

- If this property is set to the empty string, a direct connection is made to the broker identified by the HOSTNAME and PORT.
- If this property is set to a value other than the empty string, a direct connection is made to the broker through the proxy server identified by this property and the PROXYPORT property.

Data type	String
Default	Null

Proxy port:

Port number of the Web Scale proxy port.

A direct connection is made to this port on the proxy server identified by the PROXYHOSTNAME property, which forwards the connection request to the message broker. For more information, see the description of the PROXYHOSTNAME property.

Data type	Integer
Default	0

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pools:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Custom properties:

An optional set of name and value pairs for custom properties passed to WebSphere MQ.

WebSphere MQ Provider connection factory settings for application clients:

Use this panel to view or change the configuration properties of the selected connection factory for use with the MQSeries product Java Messaging Service (JMS) provider. These configuration properties control how connections are created to the associated JMS queue destination.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **JMS Providers** > *JMS provider instance*. Right click **WebSphere MQ Provider Connection Factories**, and click **New**. The following fields are displayed on the **General** tab.

A queue connection factory creates JMS connections to queue destinations. The queue connection factory is created by the MQSeries product JMS provider. A queue connection factory for the JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring MQSeries for JMS resources. For more information about configuring MQSeries product JMS resources, see the MQSeries *Using Java* book, located in the WebSphere MQ Family library.
- In MQSeries, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Name:

The required display name for the resource.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run time uses this field to retrieve configuration information. The name must match the value of the **Name** field on the General tab in the Application Client Resource Reference section of the Assembly Tool.

User:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly; for example, if the calling application uses the method createQueueConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type	String
Default	Null

Re-Enter Password:

Confirms the password.

Queue Manager:

The name of the MQSeries queue manager for this connection factory.

Connections created by this factory connect to that queue manager.

Data type	String

Host:

The name of the host on which the WebSphere MQ queue manager runs for client connection only.

Data type	String
Default	Null
Range	A valid TCP/IP host name

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type	Integer
Default	Null
Range	A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type	String
Default	Null
Range	1 through 20 ASCII characters

Transport type:

Specifies whether the WebSphere MQ client connection or JNI bindings are used for connection to the WebSphere MQ queue manager. The external JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF nonpersistent, nondurable, nontransactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type String

Default BINDINGS

Range

BINDINGS
JNI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and poses security risks that should be addressed through the use of EJB roles.

CLIENT
WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.

DIRECT
For WebSphere MQ Event Broker using DIRECT mode. DIRECT is a lightweight sockets protocol used in nontransactional, nondurable and nonpersistent Publish/Subscribe messaging. DIRECT is only works for clients and message-driven beans using the non-ASF protocol.

Recommended DIRECT is the fastest transport type and should be used where possible. Use BINDINGS when you want to satisfy additional security tasks and the queue manager is local to the JMS client.

WebSphere MQ 5.3 before CSD2 with the DIRECT setting can lose messages when used with message-driven beans and under load. This also happens with client-side based applications unless the broker's maxClientQueueSize is set to 0. You can set this to 0 with the command

```
#wempschangeproperties WAS_nodeName_server1
-e default
-o DynamicSubscriptionEngine
-n maxClientQueueSize
-v 0
-x executionGroupUUID
```

You can locate `executionGroupUUID` by starting the broker and looking in the Event Log/Applications for event 2201. This value is usually `ffffffff-0000-0000-000000000000`.
Note: The WebSphere MQ 5.3 JMS cannot be used within WAS 6.1 because WAS 6.1 has a Java 5 runtime. Therefore, cross-memory connections cannot be established with WebSphere MQ 5.3 queue managers. This can result in a performance degradation if you were previously using WebSphere MQ 5.3 and BINDINGS for your connections and move to CLIENT network connections in migrating to WAS 6.1.

Client ID:

The JMS client identifier used for connections to the MQSeries queue manager.

Data type String

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These references are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Message Retention:

Select this check box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are handled according to their disposition options.

Data type Enum
Units Not applicable
Default Cleared
Range
Selected
Unwanted messages are left on the queue.
Cleared
Unwanted messages are handled according to their disposition options.

Temporary model:

The name of the model definition used to create temporary connection factories if a connection factory does not already exist.

Data type String
Range 1 through 48 ASCII characters

Temporary queue prefix:

The prefix used for dynamic queue naming.

Data type String

Broker Control Queue:

The name of the broker control queue to which all command messages (except publications and requests to delete publications) are sent.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Queue Manager:

The name of the MQSeries queue manager that provides the Publisher and Subscriber message broker.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Pub Queue:

The name of the broker input queue that receives all publication messages for the default stream

The name of the broker's input queue (stream queue) that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Sub Queue:

The name of the broker queue from which nondurable subscription messages are retrieved.

The name of the broker queue from which nondurable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker CCSubQ:

The name of the broker queue from which nondurable subscription messages are retrieved for a ConnectionConsumer request. This property applies only for use of the Web container.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Version:

Specifies whether the message broker is provided by the MQSeries MA0C SupportPac or newer versions of WebSphere family message broker products.

Data type Enum
Default Advanced
Range
Advanced The message broker is provided by newer versions of WebSphere family message broker products (MQ Integrator and MQ Publish and Subscribe)
Basic The message broker is provided by the MQSeries MA0C SupportPac (MQSeries - Publish and Subscribe)

Cleanup level:

Specifies the level of cleanup provided by the Publish/Subscribe cleanup utility.

Data type Enum
Default SAFE
Range **ASPROP**
NONE
STRONG

Cleanup interval:

Specifies the interval, in milliseconds, between background executions of the publish/subscribe cleanup utility.

Data type Integer
Units Milliseconds
Default 6000
Range

Message selection:

Specifies where Broker message selection is performed.

Data type Enum
Default BROKER
Range

BROKER
Message selection is done at the broker location.

Message CLIENT
Message selection is done at the client location.

Publish acknowledge interval:

The interval, in number of messages, between publish requests that require acknowledgement from the broker.

Data type Integer
Default 25
Range

Sparse subscriptions:

Enables sparse subscriptions.

Data type Check box
Default Deselected

Status refresh interval:

The interval, in milliseconds, between transactions to refresh publish or subscribe status.

Data type Integer

Default 6000

Subscription store:

Specifies where WebSphere MQ stores data relating to active JMS subscriptions.

Data type Enum
Default MIGRATE
Range

MIGRATE

QUEUE

BROKER

Multicast:

Specifies whether this connection factory uses multicast transport.

Data type Enum
Default AS_CF
Range

AS_CF This connection factory uses multicast transport.

DISABLED
This connection factory does not use multicast transport.

NOT_RELIABLE
This connection factory always uses multicast transport.

RELIABLE
This connection factory uses multicast transport when the topic destination is not relable.

ENABLED
This connection factory uses reliable multicast transport.

Direct authentication:

Specifies whether to use direct broker authorization.

Data type Enum
Default NONE
Range

NONE Direct broker authorization is not used.

PASSWORD
Direct broker authorization is authenticated with a password.

CERTIFICATE
Direct broker authorization is authenticated with a certificates.

Proxy host name:

Specifies the host name of a proxy to be used for communication with WebSphere MQ.

Data type String
Range

Proxy port:

Specifies the port number of a proxy to be used for communication with WebSphere MQ.

Data type Integer
Default 0
Range

Fail if quiesce:

Specifies whether applications return from a method call if the queue manager has entered a controlled failure.

Data type Check box
Default Selected

Local server address:

Specifies the local server address.

Data type String

Polling interval:

Specifies the interval, in milliseconds, between scans of all receivers during asynchronous message delivery

Data type Integer
Units Milliseconds
Default 5000
Range

Rescan interval:

Specifies the interval in milliseconds between which a topic is scanned to look for messages that have been added to a topic out of order.

This interval controls the scanning for messages that have been added to a topic out of order with respect to a WebSphere MQ browse cursor.

Data type Integer
Units Milliseconds
Default 5000
Range

SSL cipher suite:

Specifies the cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider. The value must match the CipherSpec specified on the SVRCONN channel as the **Channel** property.

You must set this property, if you set the **SSL Peer Name** property.

SSL certificate store:

Specifies a list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. If you specify a value for this property, you must use WebSphere MQ JVM at Java 2 version 1.4.

The value is a space-delimited list of entries of the form:
`ldap://hostname:[port]`

A single slash (/) follows this value. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section “Working with Certificate Revocation Lists” in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL peer name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connection time.

If this property is not set, such certificate checking is performed.

The SSL peer name property is ignored if **SSL Cipher Suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:
`CN=QMGR.*, OU=IBM, OU=WEBSPHERE`

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the section “Distinguished Names” in the WebSphere MQ Security book.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Data type Check box

Default	Selected

WebSphere MQ queue connection factory collection

The queue connection factories configured in the WebSphere MQ messaging provider, for point-to-point messaging with JMS queues.

This panel shows a list of the WebSphere MQ queue connection factories with a summary of their configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Queue Connection Factory**. This displays a list of any existing JMS queue connection factories.

To view or change the properties of a queue connection factory, click its name in the list displayed.

To act on one or more of the connection factories listed, select the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ queue connection factory settings:

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the WebSphere MQ JMS provider. These configuration properties control how connections are created to the associated JMS queue destination.

A WebSphere MQ queue connection factory is used to create JMS connections to queues provided by WebSphere MQ for point-to-point messaging.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Queue Connection Factories**. This displays a list of any existing JMS queue connection factories.
4. Click the name of the JMS connection factory that you want to work with.

A queue connection factory for the WebSphere MQ JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book. and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at: http://www-306.ibm.com/software/integration/wmq/library/, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Component-managed authentication alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with WebSphere MQ. For example, the default Windows user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere MQ queue connection factory must specify a user ID no longer than 12 characters.
2. If you want to use Bindings transport mode on JMS queue connections to WebSphere MQ, you set the **Transport type** property to BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Container-managed authentication alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with WebSphere MQ. For example, the default Windows user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere MQ queue connection factory must specify a user ID no longer than 12 characters.
2. If you want to use Bindings transport mode on JMS queue connections to WebSphere MQ, you set the **Transport type** property to BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Mapping-configuration alias:

The module used to map authentication aliases.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum
Default Null
Range
ClientContainer
The client container maps authentication aliases.
WSLogin
The WSLogin module maps authentication aliases.
DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Host:

The name of the host on which the WebSphere MQ queue manager runs, for client connection only.

Data type String
Default Null
Range A valid TCP/IP hostname

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type Integer
Default 0
Range A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Transport type:

Whether the WebSphere MQ client connection or JNI bindings are used for connection to the WebSphere MQ queue manager.

WebSphere MQ, as the messaging provider, controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF non-persistent, non-durable, non-transactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type Enum
Units Not applicable
Default BINDINGS

Range

BINDINGS
JNI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol that can be used only when the queue manager is on the same node as the JMS client and comes at some security risks that should be addressed through the use of EJB roles.

CLIENT
WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.

Recommended

Bindings-mode offers best performance but the SYSTEM.DEF.SVRCONN must be disabled by specifying an invalid MCAUSER user ID. No other configuration is required.

Client mode uses SSL-encrypted MQI channels and certificate-based authentication using SSLPEER, and the queue manager can be located on a remote server. SSL configuration (such as certificate management) is required between WebSphere Application Server and WebSphere MQ, and J2C authentication aliases are also required.

Unless WebSphere MQ cannot be installed on the same machine as the application server, bindings mode is the logical choice. However, client mode is enabled by default so it is still necessary to disable the SYSTEM.DEF.SVRCONN channel to prevent unauthorized client access. If the MCAUSER attribute of the SYSTEM.DEF.SVRCONN channel specifies an invalid user ID, authorization is denied for all client mode access.

For additional information, see:

`http://www-128.ibm.com/developerworks/ibm/library/i-supply1i/`

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type String
Default Null
Range 1 through 20 ASCII characters

Queue manager:

The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type String
Default Null
Range A valid WebSphere MQ queue manager name, as 1 through 48 ASCII characters

Model queue definition:

The name of the model queue definition that can be used by the queue manager to create temporary queues if a queue requested does not already exist.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Client ID:

The JMS client identifier used for connections to WebSphere MQ.

Data type	String
Range	A valid JMS client ID, as ASCII characters

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

The term 'null' means leave blank; if you do this, a null value is passed and the default WebSphere MQ CCSID value is used.

Data type	String
Units	Integer
Default	Null
Range	1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multi-platform and platform-specific books Web pages; for example, at http://www-306.ibm.com/software/integration/wmq/library/, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Enable message retention:

Whether or not unwanted messages are left on the queue. If this option is not enabled, unwanted messages are dealt with according to their disposition options.

Data type	Enum
Default	Selected
Range	**Selected** Unwanted messages are left on the queue. **Cleared** Unwanted messages are dealt with according to their disposition options.

XA enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA. Enable XA if multiple resources are used in the same transaction.

If you clear this property (non-XA), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

Data type Checkbox
Default Selected
Range
Selected
The connection factory is for XA-coordination of messages
Cleared
The connection factory is for non-XA coordination of messages
Recommended Do not select to enable XA when the message queue received is the only resource in the transaction. Enable XA if transactions involve other resources, including other queues or topics.

Enable return methods during shutdown:

Whether or not applications return from a method call if the queue manager has entered a controlled shutdown.

Data type Checkbox
Default Selected
Range
Selected
Applications return from a method call if the queue manager has entered a controlled shutdown.
Cleared
Applications do not return from a method call if the queue manager has entered a controlled shutdown.

Local server address:

The local server address

If a JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can use this property to specify a port, or a range of points, that the application can bind to.

Data type String
Default Null

Range

A string in the format:

```
[ip-addr][(low-port[,high-port])]
```

For example:

- 9.20.4.98

  The channel binds to address 9.20.4.98 locally
- 9.20.4.98(1000)

  The channel binds to address 9.20.4.98 locally and uses port 1000
- 9.20.4.98(1000,2000)

  The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000
- (1000)

  The channel binds to port 1000 locally
- (1000,2000)

  The channel binds to a port in the range 1000 to 2000 locally

You can specify a host name instead of an IP address.

For direct connections, this property applies only when multicast is used and the value of the property must not contain a port number. If it does contain a port number, the connection is rejected. Therefore, the only valid values of the property are null, an IP address, or a host name.

Polling interval:

The interval, in milliseconds, between scans of all receivers during asynchronous message delivery

Data type	Integer
Units	milliseconds
Default	5000
Range	1 through 2147483647

Rescan interval:

The interval in milliseconds between which a queue is scanned to look for messages that have been added to a queue out of order.

This interval controls the scanning for messages that have been added to a queue out of order with respect to a WebSphere WebSphere MQ browse cursor.

Data type	Integer
Units	milliseconds
Default	5000
Range	1 through 2147483647

SSL Cipher Suite:

The cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the **Channel** property.

You must set this property if the **SSL Peer Name** property is to be set.

SSL CRL:

A list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. (Use of this property requires a WebSphere MQ JVM at Java 2 version 1.4.)

The value is a space-delimited list of entries of the form:
`ldap://hostname:[port]`

optionally followed by a single / (forward slash). If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section "Working with Certificate Revocation Lists" in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL Peer Name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connect-time.

The SSL Peer Name property is ignored if **SSL Cipher Suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:
`CN=QMGR.*, OU=IBM, OU=WEBSPHERE`

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the *WebSphere MQ Security* book at http://www-306.ibm.com/software/integration/wmq/library/.

Temporary queue prefix:

The prefix that is used for names of temporary JMS queues created by applications that use this connection factory.

Data type	String
Default	Null

Use connection pooling:

Whether or not to use WebSphere MQ connection pooling.

Data type	Checkbox
Default	Selected

Range

Selected	Cleared
The connection factory uses WebSphere MQ connection pooling. When a connection is no longer required, instead of destroying it, it can be pooled, and later reused. This can provide a substantial performance enhancement for repeated connections to the same queue manager.	The connection factory does not use WebSphere MQ connection pooling. When a connection is no longer required, it is destroyed. To use the same queue manager a new connection is created.

Connection pool:

An optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

WebSphere MQ topic connection factory collection

The topic connection factories configured in the WebSphere MQ messaging provider for publish/subscribe messaging with JMS topics.

This panel shows a list of the WebSphere MQ topic connection factories with a summary of their configuration properties.

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

1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Topic Connection Factory**. This displays a list of any existing queue connection factories.

To view or change the properties of a connection factory, click its name in the list displayed.

To act on one or more of the connection factories listed, select the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ topic connection factory settings:

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the WebSphere MQ as a JMS provider. These configuration properties control how connections are created to the associated JMS topic destination.

A WebSphere MQ topic connection factory is used to create JMS connections to topic destinations provided by WebSphere MQ for publish/subscribe messaging.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Topic Connection Factories**. This displays a list of any existing JMS topic connection factories.
4. Click the name of the JMS connection factory that you want to work with.

A WebSphere MQ topic connection factory has the following properties.

Note:
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book at http://www-306.ibm.com/software/integration/wmq/library/

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this topic connection factory is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the topic connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Component-managed authentication alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with WebSphere MQ. For example, the default Windows user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere MQ queue connection factory must specify a user ID no longer than 12 characters.
2. If you want to use Bindings transport mode on JMS queue connections to WebSphere MQ, you set the **Transport type** property to BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.

- Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Container-managed authentication alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Restriction:

1. User IDs longer than 12 characters cannot be used for authentication with WebSphere MQ. For example, the default Windows user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere MQ queue connection factory must specify a user ID no longer than 12 characters.
2. If you want to use Bindings transport mode on JMS queue connections to WebSphere MQ, you set the **Transport type** property to BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Mapping-configuration alias:

The module used to map authentication aliases.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum

Default Null

Range

ClientContainer
The client container maps authentication aliases.

WSLogin
The WSLogin module maps authentication aliases.

DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Host:

The name of the host on which the WebSphere MQ queue manager runs, for client connection only.

Data type	String
Default	Null
Range	A valid TCP/IP hostname

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type	Integer
Default	Null
Range	A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Transport type:

Specifies whether to use the WebSphere MQ client connection or JNI bindings for connection to the WebSphere MQ queue manager. WebSphere MQ as the JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF nonpersistent, non-durable, non-transactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type	Enum
Units	Not applicable
Default	BINDINGS

Range

BINDINGS
JNI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and comes at some security risks that should be addressed through the use of EJB roles.

CLIENT
WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.

DIRECT
For a WebSphere MQ message broker using DIRECT mode. DIRECT is a lightweight sockets protocol used in non-transactional, non-durable and non-persistent Publish/Subscribe messaging. DIRECT works only for clients and message-driven beans using the non-ASF protocol.

The type of connection to the message broker depends on the value of the PROXYHOSTNAME property, according to the following rules:
- If the PROXYHOSTNAME property is set to the empty string, a direct connection is made to the broker identified by the HOSTNAME and PORT.
- If the PROXYHOSTNAME property is set to a value other than the empty string, a direct connection is made to the broker through the proxy server identified by this property and the PROXYPORT property.

Recommended

DIRECT is the fastest transport type and should be used where possible. Use BINDINGS when you want to satisfy additional security tasks and the queue manager is local to the JMS client. QUEUED is fallback for all other cases.
Note: WebSphere MQ 5.3 before CSD2 with the DIRECT setting can lose messages when used with message-driven beans and under load. This also happens with client-side based applications unless the broker's maxClientQueueSize is set to 0. You can set this to 0 with the command `#wempschangeproperties WAS_nodeName_server1 -e default -o DynamicSubscriptionEngine -n maxClientQueueSize -v 0 -x executionGroupUUID`, where executionGroupUUID can be found by starting the broker and looking in the Event Log/Applications for event 2201. This value is usually ffffffff-0000-0000-000000000000.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type String
Default Null
Range 1 through 20 ASCII characters

Queue manager:

The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type	String
Default	Null
Range	A valid WebSphere MQ queue manager name, as 1 through 48 ASCII characters

Broker control queue:

The name of the publish/subscribe broker's control queue, to which publisher and subscriber applications send all command messages (except publications and requests to delete publications).

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Broker queue manager:

The name of the WebSphere MQ queue manager that provides the publish/subscribe message broker.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Broker publication queue:

The name of the broker's input queue (stream queue) that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	1 through 48 ASCII characters

Broker subscription queue:

The name of the broker's queue from which non-durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Broker CC subscription queue:

The name of the broker's queue from which non-durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type	String

Default Null
Range 1 through 48 ASCII characters

Broker version:

Whether the message broker is provided by the WebSphere MQ MA0C Supportpac or newer versions of WebSphere message broker products.

Data type Enum
Default Advanced
Range
Advanced
The message broker is provided by newer versions of WebSphere message broker products, such as WebSphere Business Integration Message Broker and Event Broker.
Basic The message broker is provided by the WebSphere MQ MA0C SupportPac (MQSeries - Publish/Subscribe) or MQSI working in MA0C compatibility mode.

Model queue definition:

The name of the model queue definition that the broker can use to create dynamic queues for non-default streams if the stream queue does not already exist

The name of the model queue definition that the broker can use to create dynamic queues to receive publications for streams other than the default stream. This is only used if the stream queue does not already exist. If this model queue definition does not exist, all stream queues must be defined by the administrator.

Data type String
Default Null
Range 1 through 48 ASCII characters

Enable clone support:

Select this check box to enable clone support to allow the same durable subscription across topic clones.

Data type Check box
Default Cleared
Range
Selected
Clone support is enabled.
Cleared
Clone support is disabled.

If you select this property, you must also specify a value for the **Client ID** property.

Client ID:

The JMS client identifier used for connections to WebSphere MQ.

Data type String
Range A valid JMS client ID, as ASCII characters

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

The term 'null' means leave blank; if you do this, a null value is passed and the default WebSphere MQ CCSID value is used.

Data type	String
Units	Integer
Default	Null
Range	1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www-306.ibm.com/software/integration/wmq/library/, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA. Enable XA if multiple resources are not used in the same transaction.

If you clear this property (non-XA), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

Data type	Checkbox
Default	Selected
Range	**Selected** The connection factory is for XA-coordination of messages **Cleared** The connection factory is for non-XA coordination of messages
Recommended	Do not select to enable XA when the message queue received is the only resource in the transaction. Enable XA if transactions involve other resources, including other queues or topics.

Publish/subscribe cleanup level:

The level of cleanup provided by the Publish/subscribe cleanup utility

To avoid the problems associated with non-graceful closure of subscriber objects, WebSphere MQ as a JMS provider provides a Publish/Subscribe cleanup utility that attempts to detect any earlier JMS publish/subscribe problems. If a large number of problems are detected, some performance degradation may be observed while resources are cleaned up. This utility runs transparently on a background thread and should not affect other WebSphere MQ operations.

Data type Enum
Default SAFE
Range

SAFE The Cleanup thread attempts to remove unconsumed subscription messages, or temporary queues, for failed subscriptions. This mode of cleanup does not interfere with the operation of other JMS applications.

ASPROP
The style of cleanup to use is determined by the system property com.ibm.mq.jms.cleanup, which is queried at JVM startup. This property can be set on the java command-line using the -D option, and should be set to NONE, SAFE or STRONG. Any other value causes an exception. If not set, the property defaults to SAFE. This allows easy JVM-wide change to the Cleanup level without needing to update every topic connection factory used by the system.

NONE In this special mode, no cleanup is performed; and no cleanup thread exists. Additionally, if the application is using the single-queue approach, unconsumed messages can be left on the queue.

This option can be useful if the application is distant from the queue manager, and especially if it only publishes rather than subscribes. However, some application should perform cleanup on the queue manager to deal with any unconsumed messages - this could be a JMS application with CLEANUP(SAFE) or CLEANUP(STRONG), or the WebSphere MQ manual cleanup utility.

STRONG
The cleanup thread performs as CLEANUP(SAFE), but also clears the SYSTEM.JMS.REPORT.QUEUE of any unrecognized messages.

Publish/subscribe cleanup interval:

The interval, in milliseconds, between background executions of the publish/subscribe cleanup utility.

Data type Integer
Default 60000
Range 1 through 2147483647

Message selection:

Whether message selection is done at the broker or client.

Data type Enum
Default BROKER
Range

BROKER
Message selection is done at the broker.

CLIENT
Message selection is done at the client.

Publish acknowledgement interval:

The interval, in number of messages, between publish requests that require acknowledgement from the broker.

Data type Integer
Default 25
Range 1 through 2147483647

Enable sparse subscriptions:

Select this option to support subscriptions that receive infrequent matching messages.

Data type Checkbox
Default Cleared
Range

Selected
Subscriptions can receive infrequent matching messages. This value requires that the subscription queue can be opened for browse.

Cleared
Sparse subscriptions are not supported. Subscriptions receive frequent matching messages.

Publish/subscribe status interval:

The interval, in milliseconds, between transactions to refresh publish/subscribe status.

Data type Integer
Default 60000
Range 1 through 2147483647

Persistent subscriptions store:

Where WebSphere MQ stores persistent data relating to active JMS subscriptions.

Data type Enum
Default MIGRATE

Range

MIGRATE
This option dynamically selects the queue-based or broker-based subscription store based on the levels of queue manager and publish/subscribe broker installed. If both queue manager and broker are capable of supporting SUBSTORE(BROKER), this behaves as SUBSTORE(BROKER); otherwise it behaves as SUBSTORE(QUEUE). Additionally, SUBSTORE(MIGRATE) transfers durable subscription information from the queue-based subscription store to the broker-based store.

QUEUE
Subscription information is stored on SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE on the local queue manager.

BROKER
Subscription information is stored by the publish/subscribe broker used by the application. This option requires recent levels of queue manager and publish/subscribe broker. This subscription store requires recent levels of both queue manager and publish/subscribe broker. It is designed to provide improved resilience.

Enable multicast transport:

Whether or not this connection factory uses multicast transport.

With multicast, messages are delivered to all consumers. This is useful in environments where there are a large number of clients that all want to receive the same messages, because with multicast only one copy of each message is sent. Multicast reduces the total amount of network traffic. Reliable multicast is standard multicast with a reliability layer added.

Data type
Enum

Default
NOTUSED

Range

NOTUSED
This connection factory does not use multicast transport.

ENABLED
This connection factory uses multicast transport, but does not provide a reliable multicast connection.

ENABLED_IF_AVAILABLE
This connection factory uses multicast transport if the message broker supports it.

ENABLED_RELIABLE
This connection factory uses reliable multicast transport

ENABLED_RELIABLE_IF_AVAILABLE
This connection factory uses reliable multicast transport if the message broker supports it.

Direct broker authentication:

Whether the broker uses basic or certificate-based authentication for direct connections.

This property selects the authentication on a direct connections (if the TRANSPORT property is set to DIRECT).

Data type Enum
Default NONE
Range

NONE Direct broker authentication is not used.

PASSWORD
Password-based authentication is used for direct connections. Authentication is performed based on a user ID and password provided by an authentication alias. The authentication alias used is obtained from one of the following properties:

- Component-managed Authentication Alias, for application-managed authentication.
- Container-managed Authentication Alias, for container-managed authentication.

CERTIFICATE
Certificate-based authentication is used for direct connections. The SSLPEERNAME and SSLCRL properties are used to perform the authentication checks.

You can use certificate-based authentication when connecting directly to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker.

Proxy host name:

Host name of the Web Scale proxy host.

A direct connection is made to the proxy server, which forwards the connection request to the message broker.

If the TRANSPORT property is set to DIRECT, the type of connection to the message broker depends on the value of this property, according to the following rules:

- If this property is set to the empty string, a direct connection is made to the broker identified by the HOSTNAME and PORT.
- If this property is set to a value other than the empty string, a direct connection is made to the broker through the proxy server identified by this property and the PROXYPORT property.

Data type String
Default Null

Proxy port:

Port number of the Web Scale proxy port.

A direct connection is made to this port on the proxy server identified by the PROXYHOSTNAME property, which forwards the connection request to the message broker. For more information, see the description of

the PROXYHOSTNAME property.

Data type Integer
Default 0

Enable return methods during shutdown:

Whether or not applications return from a method call if the queue manager has entered a controlled shutdown.

Data type Checkbox
Default Selected
Range
Selected
Applications return from a method call if the queue manager has entered a controlled shutdown.
Cleared
Applications do not return from a method call if the queue manager has entered a controlled shutdown.

Local server address:

The range of local ports to be used when making a connection to a WebSphere MQ queue manager

If a JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can use this property to specify a port, or a range of points, that the application can bind to.

Data type String
Default Null

Range

A string in the format:

```
[ip-addr][(low-port[,high-port])]
```

For example:

- 9.20.4.98

  The channel binds to address 9.20.4.98 locally
- 9.20.4.98(1000)

  The channel binds to address 9.20.4.98 locally and uses port 1000
- 9.20.4.98(1000,2000)

  The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000
- (1000)

  The channel binds to port 1000 locally
- (1000,2000)

  The channel binds to a port in the range 1000 to 2000 locally

You can specify a host name instead of an IP address.

For direct connections, this property applies only when multicast is used and the value of the property must not contain a port number. If it does contain a port number, the connection is rejected. Therefore, the only valid values of the property are null, an IP address, or a host name.

Polling interval:

The interval, in milliseconds, between scans of all receivers during asynchronous message delivery

Data type	Integer
Units	milliseconds
Default	5000
Range	1 through 2147483647

Rescan interval:

The interval in milliseconds between which a topic is scanned to look for messages that have been added to a topic out of order.

This interval controls the scanning for messages that have been added to a topic out of order with respect to a WebSphere MQ browse cursor.

Data type	Integer
Units	milliseconds
Default	5000
Range	1 through 2147483647

SSL cipher suite:

The cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the **Channel** property.

You must set this property if the **SSL Peer Name** property is to be set.

SSL CRL:

A list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. (Use of this property requires a WebSphere MQ JVM at Java 2 version 1.4.)

The value is a space-delimited list of entries of the form:
`ldap://hostname:[port]`

optionally followed by a single / (forward slash). If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section "Working with Certificate Revocation Lists" in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL peer name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connect-time.

The SSL Peer Name property is ignored if **SSL Cipher Suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:
`CN=QMGR.*, OU=IBM, OU=WEBSPHERE`

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the *WebSphere MQ Security* book; for example, the section "Distinguished Names" at http://www-306.ibm.com/software/integration/wmq/library/.

Enable MQ Connection Pooling:

Whether or not to use WebSphere MQ connection pooling.

Data type Checkbox

Default Selected

Range

Selected	**Cleared**
The connection factory uses WebSphere MQ connection pooling. When a connection is no longer required, instead of destroying it, it can be pooled, and later reused. This can provide a substantial performance enhancement for repeated connections to the same queue manager.	The connection factory does not use WebSphere MQ connection pooling. When a connection is no longer required, it is destroyed. To use the same queue manager a new connection is created.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pools:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

WebSphere MQ queue destination collection

The queue destinations configured in the WebSphere MQ messaging provider for point-to-point messaging with JMS queues.

This panel shows a list of the WebSphere MQ queue destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Queue Destinations**. This displays a list of any existing JMS queue destinations.

To create a new queue destination, click **New**.

To browse or change the properties of a queue destination, select its name in the list displayed.

To act on one or more of the queue destinations listed, click the check box next to the name of the queue, then use the buttons provided.

WebSphere MQ queue settings:

Use this panel to browse or change the configuration properties of the selected JMS queue destination for point-to-point messaging with WebSphere MQ as a messaging provider.

A WebSphere MQ queue destination is used to configure the properties of a JMS queue. Connections to the queue are created by the associated JMS queue connection factory for WebSphere MQ as a messaging provider.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **Messaging Providers** → **WebSphere MQ**.

2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Queue Destinations**. This displays a list of any existing JMS queue destinations.
4. Click the name of the JMS queue destination that you want to work with.

A queue for use with the WebSphere MQ JMS provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The **WebSphere MQ Queue Connection Properties** group of fields in this pane contains five properties fields:
  - queue manager host
  - queue manager port
  - server connection channel name
  - user ID
  - password.

  Set these properties if you want to use the MQ Config additional property feature for managing the physical WebSphere MQ queue on the external WebSphere MQ queue manager. For additional information, see the related link to WebSphere MQ queue settings (MQ Config) at the end of this topic.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the queue into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String
Default Null

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default Application defined
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Non persistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Default Application defined

Range

Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.

Queue defined
Messages on the destination have their persistence defined by the WebSphere MQ destination definition properties.

Specified
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority you must define a priority on the **Specified Priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this destination, in the range 0 (lowest) through 9 (highest)

Data type Integer
Units Message priority level
Default 0
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this destination is defined by the application or the **Specified Expiry** property, or messages on the destination never expire (have an unlimited expiry timeout)

Data type Enum
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
The expiry timeout for messages on this destination is defined by the application that put them onto the destination.

SPECIFIED
The expiry timeout for messages on this destination is defined by the **Specified Expiry** property.*If you select this option, you must define a timeout on the **Specified Expiry** property.*

UNLIMITED
Messages on this destination have no expiry timeout, so those messages never expire.

Specified expiry:

If the **Expiry Timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this destination expire.

Data type Integer
Units Milliseconds
Default 0
Range Greater than or equal to 0
- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Base queue name:

The name of the queue to which messages are sent, on the queue manager specified by the **Base Queue Manager Name** property.

Data type	String
Default	Null
Range	1 through 48 ASCII characters

Base queue manager name:

The name of the WebSphere MQ queue manager to which messages are sent

This queue manager provides the queue specified by the **Base Queue Name** property.

Data type	String
Units	En_US ASCII characters
Default	Null
Range	A valid WebSphere MQ Queue Manager name, as 1 through 48 ASCII characters

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type	String
Units	Integer
Default	Null
Range	1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www-3.ibm.com/software/integration/wmq/library, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Use native encoding:

Whether or not the destination should use native encoding (appropriate encoding values for the Java platform).

Data type	Check box
Default	Cleared
Range	**Cleared** Native encoding is not used, so specify the properties below for integer, decimal, and floating point encoding. **Selected** Native encoding is used (to provide appropriate encoding values for the Java platform). For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type Enum
Units Not applicable
Default NORMAL
Range
NORMAL
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

If native encoding is not enabled, select the type of floating point encoding.

Data type Enum
Default IEEENORMAL
Range
IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Target client:

Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application

Data type Enum
Default JMS
Range
JMS The target is a JMS-compliant application.
MQ The target is a non-JMS, traditional WebSphere MQ application.

Queue manager host:

The name of host for the queue manager on which the queue destination is created.

Data type	String
Default	Null
Range	A valid TCP/IP hostname

Queue manager port:

The number of the port used by the queue manager on which this queue is defined.

Data type	String
Units	A valid TCP/IP port number.
Default	Null
Range	A valid TCP/IP port number. This port must be configured on the WebSphere MQ queue manager.

Server connection channel name:

The name of the channel used for connection to the WebSphere MQ queue manager.

Data type	String
Default	Null
Range	1 through 20 ASCII characters

User name:

The user ID used, with the **Password** property, for authentication when connecting to the queue manager to define the queue destination.

*If you specify a value for the **User name** property, you must also specify a value for the **Password** property.*

Data type	String
Default	Null

Password:

The password, used with the **User name** property, for authentication when connecting to the queue manager to define the queue destination.

*If you specify a value for the **User name** property, you must also specify a value for the **Password** property.*

Data type	String
Default	Null

WebSphere MQ queue settings (MQ Config):

Use this panel to browse or change the configuration properties defined to WebSphere MQ for the selected queue destination.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.

2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Queue Destinations**. This displays a list of any existing JMS queue destinations.
4. Click the name of the JMS queue destination that you want to work with.
5. Under Additional Resources, click **MQ Config**.

A WebSphere MQ queue destination is used to configure the properties of a JMS queue. A queue for use with the WebSphere MQ JMS provider has the following extra properties defined to WebSphere MQ.
Notes

Note:

- To be able to browse or change these MQ Config properties, the WebSphere MQ Queue Manager on which the queue resides must be configured for remote administration and be running. You must also have installed the WebSphere MQ client. If you have not done this, the administrative console displays messages like the following:

  ```
  The WMQQueueDefiner MBean has encountered an error.
  WMSG0331E: The MQ Client is required for this functionality, but it is not installed.
  ```
- These MQ Config properties can be used only to view or change the properties of local queues. You cannot use MQ Config to administer alias or remote queues.
- Some properties displayed are read-only and cannot be changed.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ JMS resources. For more information about configuring WebSphere MQ JMS resources, see the *WebSphere MQ: Using Java* book; for example from the WebSphere MQ multiplatform library Web page at http://www.ibm.com/software/ts/mqseries/library/manualsa/manuals/crosslatest.html.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Base Queue Name:

The name of the local queue to which messages are sent, on the queue manager specified by the **Base Queue Manager Name** property.

Data type String

Base Queue Manager Name:

The name of the WebSphere MQ queue manager to which messages are sent.

This queue manager provides the queue specified by the **Base Queue Name** property.

Data type String

Queue Manager Host:

The name of host for the queue manager on which the queue destination is created.

Data type String

Queue Manager Port:

The number of the port used by the queue manager on which this queue is defined.

Data type Integer
Range A valid TCP/IP port number. This port must be configured on the WebSphere MQ queue manager.

Server Connection Channel Name:

The name of the channel used for connection to the WebSphere MQ queue manager.

Data type String
Range 1 through 20 ASCII characters

User ID:

The user ID used, with the **Password** property, for authentication when connecting to the queue manager to define the queue destination.

*If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.*

Data type String

Password:

The password, used with the **User ID** property, for authentication when connecting to the queue manager to define the queue destination.

*If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.*

Data type String

Description:

The WebSphere MQ queue description, for administrative purposes within WebSphere MQ.

Data type String
Default Null
Range 1 through 64 ASCII characters.

Inhibit Put:

Whether or not put operations are allowed for this queue.

Data type Enum
Default Put Inhibited
Range
Put Allowed
Put operations are allowed for this queue.
Put Inhibited
Put operations are not allowed for this queue.

Persistence:

Whether messages on the queue are persistent or non-persistent.

Data type Enum

Default Persistent

Range

Persistent
Messages on the queue are persistent.

Non persistent
Messages on the queue are not persistent.

Cluster Name:

The name of the cluster to which the WebSphere MQ queue manager belongs.

If you specify a value for **Cluster Name**, you should not specify a value for **Cluster Name List**. Cluster names must conform to the rules described in the *WebSphere MQ MQSC Command Reference* book.

Data type String

Default Null

Range A valid WebSphere MQ name for a queue manager cluster, as 1 through 48 ASCII characters

Cluster Name List:

The name of the cluster namelist to which the WebSphere MQ queue manager belongs.

If you specify a value for **Cluster Name**, you should not specify a value for **Cluster Name List**. Cluster names must conform to the rules described in the *WebSphere MQ MQSC Command Reference* book.

Data type String

Default Null

Range A valid WebSphere MQ name for a list of queue manager clusters, as 1 through 48 ASCII characters

Default Binding:

The default binding to be used when the queue is defined as a cluster queue.

Data type Enum

Default On Open

Range

On Open
The queue handle is bound to a specific instance of the cluster queue when the queue is opened.

Not Fixed
The queue handle is not bound to any particular instance of the cluster queue. This allows the queue manager to select a specific queue instance when the message is put, and to change that selection subsequently should the need arise.

Inhibit Get:

Whether or not get operations are allowed for this queue.

Data type Enum

Default Get Inhibited

Range

Get Inhibited
Get operations are not allowed for this queue.
Get Allowed
Get operations are allowed for this queue.

Maximum Queue Depth:

The maximum number of messages allowed on the queue.

Data type Integer
Units Number of messages
Default 0
Range A value greater than or equal to zero, and less than or equal to 640 000.

For more information about the maximum value allowed, see the *WebSphere MQ MQSC Command Reference*.

If this value is reduced, any messages that are already on the queue are not affected, even if the number of messages exceeds the new maximum.

Maximum Message Length:

The maximum length, in bytes, of messages on this queue.

Data type Integer
Units Number of bytes
Default 0
Range A value greater than or equal to zero, and less than or equal to the maximum message length for the queue manager and WebSphere MQ platform. For more information about the maximum value allowed, see the *WebSphere MQ MQSC Command Reference*.

If this value is reduced, any message that is already on the queue are not affected, even if the message length exceeds the new maximum.

Shareability:

Whether multiple applications can get messages from this queue.

Data type Enum
Default Shareable
Range

Shareable
More than one application instance can get messages from the queue.
Not Shareable
Only one application instance can get messages from the queue.

Input Open Option:

The default share option for applications opening this queue for input

Data type Enum
Default Exclusive
Range
Exclusive
The open request is for exclusive input from the queue.
Shared
The open request is for shared input from the queue.

Message Delivery Sequence:

The order in which messages are delivered from the queue in response to get requests.

Data type Enum
Default FIFO
Range
FIFO Messages are delivered in first in first out (FIFO) order. Priority is ignored for messages on this queue.
Priority
Messages are delivered in first-in-first-out (FIFO) order within priority. This is the default supplied with WebSphere MQ, but your installation might have changed it.

Backout Threshold:

The maximum number of times that a message can be backed out. If this threshold is reached, the message is requeued on the backout queue specified by the **Backout Requeue Name** property.

The WebSphere MQ queue manager keeps a record of the number of times that each message has been backed out. When this number reaches a configurable threshold, the connection consumer requeues the message on a named backout queue. If this requeue fails for any reason, the message is removed from the queue and either requeued to the dead-letter queue, or discarded.

Data type Integer
Default 0
Range
0 Never requeue messages
1 or more
The number of times that a message has been backed, at which the message is requeued on a named backout queue.

Backout Requeue Name:

The name of the backout queue to which messages are requeued if they have been backed out more than the backout threshold.

The WebSphere MQ queue manager keeps a record of the number of times that each message has been backed out. When this number reaches a configurable threshold, the connection consumer requeues the message on a named backout queue. If this requeue fails for any reason, the message is removed from the queue and either requeued to the dead-letter queue, or discarded.

Data type String
Default Null
Range 1 through 48 characters.

Harden Get Backout:

Whether hardening should be used to ensure that the count of the number of times that a message has been backed out is accurate.

Data type	Enum
Default	Hardened
Range	**Hardened** The count is hardened. **Not Hardened** The count is not hardened. This is the default supplied with WebSphere MQ, but your installation might have changed it.

Default Priority:

The default message priority for this destination, used if no priority is provided with a message.

Data type	Integer
Default	0
Range	0 (lowest priority) through 9 (highest priority)

WebSphere MQ topic destination collection

The JMS topic destinations configured in the WebSphere MQ messaging provider for point-to-point messaging with JMS topics. Use this panel to create or delete topic destinations, or to select a topic destination to view or change its configuration properties.

This panel shows a list of JMS topic destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Topic Destinations**. This displays a list of any existing JMS topic destinations.

To create a new topic destination, click **New**.

To view or change the properties of a topic destination, select its name in the list displayed.

To act on one or more of the topic destinations listed, click the check box next to the name of the topic, then use the buttons provided.

WebSphere MQ topic settings:

Use this panel to browse or change the configuration properties of the selected JMS topic destination for publish/subscribe messaging with WebSphere MQ as a messaging provider.

A WebSphere MQ topic destination is used to configure the properties of a JMS topic for WebSphere MQ as a messaging provider. Connections to the topic are created by the associated topic connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.

2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, under Additional Resources, click **WebSphere MQ Topic Destinations**. This displays a list of any existing JMS topic destinations.
4. Click the name of the JMS topic destination that you want to work with.

A WebSphere MQ topic has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the topic is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the topic into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the destination.

QUEUE DEFINED
Messages on the destination have their persistence defined by the WebSphere MQ destination definition properties.

PERSISTENT
Messages on the destination are persistent.

NON PERSISTENT
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified Priority** property

Data type Enum
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.

QUEUE DEFINED
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

SPECIFIED
The priority of messages on this destination is defined by the **Specified Priority** property. *If you select this option, you must define a priority on the **Specified Priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this destination, in the range 0 (lowest) through 9 (highest)

Data type Integer
Units Message priority level
Default 0
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this destination is defined by the application or the **Specified Expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type Enum
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
The expiry timeout for messages on this destination is defined by the application that put them onto the destination.
SPECIFIED
The expiry timeout for messages on this destination is defined by the **Specified Expiry** property. *If you select this option, you must define a timeout on the **Specified Expiry** property.*
UNLIMITED
Messages on this destination have no expiry timeout, so those messages never expire.

Specified expiry:

If the **Expiry Timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this destination expire.

Data type Integer
Units Milliseconds
Default 0
Range Greater than or equal to 0
- 0 indicates that messages never timeout
- Other values are an integer number of milliseconds

Base topic name:

The name of the WebSphere MQ topic to which messages are sent.

Data type String
Range Depends on the broker used. For details, see the documentation for your broker; for example the *WebSphere MQ Event Broker* library at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/wsmqebv21.html.

CCSID:

The coded character set identifier for use with WebSphere MQ.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String
Units Integer
Default Null
Range 1 through 65535

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Use native encoding:

Whether or not the destination should use native encoding (appropriate encoding values for the Java platform).

Data type Check box
Units Not applicable
Default Cleared
Range
Cleared
Native encoding is not used, so specify the properties below for integer, decimal, and floating point encoding.
Selected
Native encoding is used (to provide appropriate encoding values for the Java platform).

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type Enum
Default NORMAL

Range

NORMAL
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

If native encoding is not enabled, select the type of floating point encoding.

Data type Enum
Default IEEENORMAL
Range

IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Target client:

Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application

Data type Enum
Default JMS
Range

JMS The target is a JMS-compliant application.
MQ The target is a non-JMS, traditional WebSphere MQ application.

Broker durable subscription queue:

The name of the broker's queue from which durable subscription messages are retrieved

The subscriber specifies the name of the queue when it registers a subscription.

Data type String
Default Null
Range 1 through 48 ASCII characters

Broker CC durable subscription queue:

The name of the broker's queue from which durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type String
Default Null
Range 1 through 48 ASCII characters

Enable multicast:

Whether or not this topic destination uses multicast transport.

With multicast, messages are delivered to all consumers. This is useful in environments where there are a large number of clients that all want to receive the same messages, because with multicast only one copy of each message is sent. Multicast reduces the total amount of network traffic. Reliable multicast is standard multicast with a reliability layer added.

Data type Enum

Default NOTUSED

Range

NOTUSED
This destination does not use multicast transport.

ENABLED
This destination uses multicast transport, but does not provide a reliable multicast connection.

ENABLED_IF_AVAILABLE
This destination uses multicast transport if the message broker supports it.

ENABLED_RELIABLE
This destination uses reliable multicast transport

ENABLED_RELIABLE_IF_AVAILABLE
This destination uses reliable multicast transport if the message broker supports it.

Configuring JMS resources for the WebSphere MQ messaging provider

Use the following tasks to configure the connection factories and destinations for the WebSphere MQ JMS provider.

You only need to complete these tasks if WebSphere Application Server supports enterprise applications that use JMS resources provided by WebSphere MQ. To enable use of resources provider by WebSphere MQ, you must have installed and configured WebSphere MQ JMS support.

You can use the WebSphere administrative console to configure JMS connections factories, JMS queues, and JMS topics for WebSphere MQ as the messaging provider.

Using the administrative console, if you set the scope of the WebSphere MQ messaging provider to cell scope or to node scope for a WebSphere Application Server version 6 node, you can configure JMS 1.1 resources and properties. This includes unified JMS connection factories for use by both point-to-point and publish/subscribe JMS 1.1 applications. With JMS 1.1, this approach is preferred to the domain-specific queue connection factory and topic connection factory. If you set the scope to a WebSphere Application Server Version 5 node, you can only configure domain-specific JMS resources, and the subset of properties that apply to WebSphere Application Server Version 5.

For more information about configuring JMS resources for the WebSphere MQ messaging provider, see the following topics. These topics include optional steps for you to create a new JMS resource.

Configuring resources for WebSphere Application Server version 6:
- Configuring a unified JMS connection factory
- Configuring a JMS queue connection factory
- Configuring a JMS topic connection factory
- Configuring a JMS queue
- Configuring a JMS topic
- Enabling WebSphere MQ JMS connection pooling

Configuring a unified JMS connection factory, for WebSphere MQ

Use this task to browse or configure a unified JMS connection factory for use with WebSphere MQ as a messaging provider. This task contains an optional step for you to create a new unified JMS connection factory.

This topic describes how to configure a unified JMS connection factory for WebSphere MQ as a messaging provider on an Application Server version 6 node. With JMS 1.1, this approach is preferred to the domain-specific JMS queue connection factory and JMS topic connection factory.

If you want to configure a JMS queue connection factory or topic factory, see the related tasks.

To browse or configure a unified JMS connection factory for use with WebSphere MQ as a messaging provider, use the administrative console to complete the following steps:

1. Display the WebSphere MQ messaging provider. In the navigation pane, expand **Resources** → **JMS Providers** → **WebSphere MQ**.
2. **Optional:** Change the **Scope** setting to the level at which the connection factory is visible to applications.
3. In the contents pane, under Additional Properties, click **WebSphere MQ Connection Factories** This displays a table listing any existing unified JMS connection factories, with a summary of their properties.
4. To browse or change the properties of an existing unified JMS connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this connection factory is known for administrative purposes within IBM WebSphere Application Server.

      JNDI name
      The JNDI name that is used to bind the connection factory into the name space.

      CCSID
      The coded character set identifier for use with the WebSphere MQ queue manager; for example: `850`.

   c. Click **Apply**. This defines the JMS connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
5. **Optional:** Change properties for the unified JMS connection factory, according to your needs.
6. **Optional:** Change connection pool properties and session pool properties, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS queue connection factory for WebSphere MQ

Use this task to browse or change a JMS queue connection factory for use with WebSphere MQ as a messaging provider. This task contains an optional step for you to create a new JMS queue connection factory.

With JMS 1.1, the preferred approach is to use unified JMS connection factories instead of the domain-specific JMS queue connection factory and JMS topic connection factory. If you want to configure a unified JMS connection factory, see “Configuring a unified JMS connection factory, for WebSphere MQ.”

To browse or configure a JMS queue connection factory for use with WebSphere MQ as a messaging provider, use the administrative console to complete the following steps:

1. Display the WebSphere MQ messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. Select the WebSphere MQ provider for which you want to configure a queue connection factory.
3. **Optional:** Change the **Scope** setting to the level at which the connection factory is visible to applications.
4. In the contents pane, under Additional Properties, click **Queue connection factories** This displays a table listing any existing JMS queue connection factories, with a summary of their properties.
5. To browse or change an existing JMS queue connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this connection factory is known for administrative purposes within IBM WebSphere Application Server.

      JNDI name
      The JNDI name that is used to bind the connection factory into the name space.

      CCSID
      The coded character set identifier for use with the WebSphere MQ queue manager; for example: `850`.
   c. Click **Apply**. This defines the connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
6. **Optional:** Change properties for the queue connection factory, according to your needs.
7. **Optional:** Change connection pool properties and session pool properties, according to your needs.
8. Click **OK**.
9. Save any changes to the master configuration.
10. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS topic connection factory for WebSphere MQ

Use this task to browse or configure a JMS topic connection factory for publish/subscribe messaging with WebSphere MQ as a messaging provider. This task contains an optional step for you to create a new JMS topic connection factory.

With JMS 1.1, the preferred approach is to use unified JMS connection factories instead of the domain-specific JMS queue connection factory and JMS topic connection factory. If you want to configure a unified JMS connection factory, see “Configuring a unified JMS connection factory, for WebSphere MQ” on page 957.

To configure a topic connection factory for use with the WebSphere MQ as a messaging provider, use the administrative console to complete the following steps:

1. Display the WebSphere MQ messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. Select the WebSphere MQ provider for which you want to configure a topic connection factory.
3. **Optional:** Change the **Scope** setting to the level at which the connection factory is visible to applications.
4. In the contents pane, under Additional Properties, click **Topic connection factories** This displays a table listing any existing JMS topic connection factories, with a summary of their properties.
5. To browse or change an existing JMS topic connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.

b. Specify the following required properties. You can specify other properties, as described in a later step.

Name The name by which this destination is known for administrative purposes within IBM WebSphere Application Server.

JNDI Name
The JNDI name that is used to bind the destination into the name space.

CCSID
The coded character set identifier for use with the WebSphere MQ queue manager; for example: `850`.

c. Click **Apply**. This defines the destination to WebSphere Application Server, and enables you to browse or change additional properties.

6. **Optional:** Change properties for the topic connection factory, according to your needs.
7. **Optional:** Change connection pool properties and session pool properties, according to your needs.
8. Click **OK**.
9. Save any changes to the master configuration.
10. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS queue destination for WebSphere MQ

Use this task to browse or change a JMS queue destination for point-to-point messaging with WebSphere MQ as a messaging provider. This task contains an optional step for you to create a new JMS queue destination.

To browse or configure a JMS queue destination for use with WebSphere MQ as a messaging provider, use the administrative console to complete the following steps:

1. Display the WebSphere MQ messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. Select the WebSphere MQ provider for which you want to configure a queue destination.
3. **Optional:** Change the **Scope** setting to the level at which the JMS destination is visible to applications.
4. In the contents pane, under Additional Properties, click **Queues** This displays a table listing any existing JMS queue destinations, with a summary of their properties.
5. To browse or change the properties of an existing JMS queue destination, click its name in the list. Otherwise, to create a new queue, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this queue destination is known for administrative purposes within IBM WebSphere Application Server.

      JNDI name
      The JNDI name that is used to bind the queue destination into the name space.

      Base Queue Name
      The name of the queue to which messages are sent, on the queue manager specified by the **Base Queue Manager Name** property.

      CCSID
      The coded character set identifier for use with the WebSphere MQ queue manager; for example: `850`.

   c. Click **Apply**. This defines the queue destination to WebSphere Application Server, and enables you to browse or change additional properties.
6. **Optional:** Change properties for the queue destination, according to your needs.

7. **Optional:** If you want WebSphere Application Server to try to use the WebSphere MQ queue manager's remote administration utilities to create the queue, configure the WebSphere MQ Queue Connection properties.

   If you have already created your underlying queue in WebSphere MQ using its administration tools (such as runmqsc or MQ Explorer), you do not need to configure any of the WebSphere MQ Queue Connection properties. You only need to configure these properties if you want WebSphere Application Server to try to use the WebSphere MQ queue manager's remote administration utilities to create the queue.

   To be able to browse or change these MQ Config properties, you must have installed the WebSphere MQ client. If you have not done this, the administrative console displays messages like the following:

   ```
   The WMQQueueDefiner MBean has encountered an error.
   WMSG0331E: The MQ Client is required for this functionality, but it is not installed.
   ```

   Note: For any changes to these properties to take effect on the queue manager, the WebSphere MQ Queue Manager on which the queue resides (or will reside) must be configured for remote administration and be running.

   For more details about these properties, see WebSphere MQ config properties for the queue destination.
8. Click **OK**.
9. Save any changes to the master configuration.
10. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS topic destination for WebSphere MQ

Use this task to browse or change a JMS topic destination for publish/subscribe messaging with WebSphere MQ as a messaging provider. This task contains an optional step for you to create a new JMS topic destination.

To configure a JMS topic destination for use with WebSphere MQ as a messaging provider, use the administrative console to complete the following steps:

1. Display the WebSphere MQ messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
2. Select the WebSphere MQ provider for which you want to configure a topic destination.
3. **Optional:** Change the **Scope** setting to the level at which the JMS destination is visible to applications.
4. In the contents pane, under Additional Properties, click **Topics** This displays a table listing any existing JMS topic destinations, with a summary of their properties.
5. To browse or change an existing JMS topic destination, click its name in the list. Otherwise, to create a new topic destination, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this connection factory is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the connection factory into the name space.

      Base Topic Name
      The name of the WebSphere MQ topic to which messages are sent.

      CCSID
      The coded character set identifier for use with the WebSphere MQ queue manager; for example: `850`.
   c. Click **Apply**. This defines the topic destination to WebSphere Application Server, and enables you to browse or change additional properties.

6. **Optional:** Change properties for the topic destination, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring WebSphere MQ connection pooling

Use this task to browse or change properties of WebSphere MQ connection pooling for JMS connections from an application server to WebSphere MQ as a JMS provider.

To enable WebSphere MQ connection pooling for an application server, use the administrative console to complete the following steps:

1. Display the Message Listener Service properties for the application server
   a. In the navigation pane, click **Servers** → **Application Servers**
   b. In the content pane, click the name of the application server.
   c. Under Additional Properties, click **Message Listener Service properties**.
2. Select Custom Properties, to enable WebSphere MQ connection pooling, add the following custom properties:
   mqjms.pooling.threshold
   The maximum number of unused connections in the pool.
   mqjms.pooling.timeout
   The timeout in milliseconds for unused connections in the pool.
3. Click **OK**.
4. Save any changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the application server.

For additional information, see the books in the WebSphere MQ library
`http://www-306.ibm.com/software/integration/wmq/library/`

WebSphere MQ JMS connection pooling:

To improve the overall performance of JMS within the system, the message listener service enables the connection pooling facility provided by the WebSphere MQ JMS implementation.

This support for connection pooling does not affect the performance of a message listener, because it retains its connections while listening on a destination, but does affect the overall JMS system performance. When a connection is no longer required, WebSphere MQ can pool the connection then reuse it later instead of destroying it.

Note: This support is only available for use with WebSphere MQ as a JMS provider.

To enable WebSphere MQ connection pooling and configure the characteristics of the WebSphere MQ connection pool, see Enabling WebSphere MQ JMS connection pooling.

Configuring custom properties for the WebSphere MQ messaging provider

You can use WebSphere MQ as a messaging provider for WebSphere Application Server. Many of the properties for a WebSphere MQ messaging provider can be selected from panels in the WebSphere Application Server administration console, but there are further properties that you can configure as custom properties. Use this task to create custom properties for destinations and connection factories when using WebSphere MQ as a messaging provider.

To create a new custom property for a destination or for a connection factory when using WebSphere MQ as a messaging provider, use the administrative console to complete the following steps:

1. Display the WebSphere MQ messaging provider. In the navigation pane, expand **Resources** → **JMS**.

2. Select the type of JMS resource for which you want create custom properties.
3. **Optional:** Change the **Scope** setting to the level at which the resource definition is visible to applications.
4. In the contents pane, select the specific JMS resource name. This displays information about the resource.
5. To create custom properties, under Additional Properties, select **Custom properties** .
   a. Click **New** in the content pane.
   b. Specify the following properties.

      Name The name of the custom property. This property is required.

      Value The value for the custom property.

      Type The type of the custom property. Select the custom property type from the list.
   c. Click **Apply**. This defines the custom property to WebSphere Application Server, and enables you to browse or change additional properties.
6. Click **OK**.
7. Save any changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

WebSphere MQ custom properties:

WebSphere Application Server supports the use of custom properties to define WebSphere MQ properties. This is useful because it enables WebSphere Application Server to work with later versions of WebSphere MQ which might have properties that are not exposed in the WebSphere Application Server administrative console.

For instructions on how to create a new custom property, see “Configuring custom properties for the WebSphere MQ messaging provider” on page 961.

In WebSphere Application Server, Version 6.1, the custom properties that you define are validated by the WebSphere MQ client jar files contained in WebSphere Application Server. In previous versions, this was done within WebSphere Application Server itself, and then by the WebSphere MQ client jar files. If you have defined a property that is not valid for WebSphere MQ, the WebSphere MQ client jar files create an exception, which is caught by WebSphere Application Server, and logged in the Systemout.log and SystemErr.log. Examples of error messages are given at the end of this topic.

When a later version of WebSphere MQ is available that is supported by the WebSphere Application Server installation, new MQ properties might be created that are not known to WebSphere Application Server. You can configure these as custom properties through WebSphere Application Server so that they are recognized by the WebSphere MQ client jars. You can also configure WebSphere Application Server to point to the WebSphere MQ client jars in the external JMS provider, as described in “Installing and configuring WebSphere MQ as a JMS provider” on page 884.

For information on valid values for WebSphere MQ properties, refer to *WebSphere MQ Using Java* or *WebSphere MQ System Administration*, which are available from the IBM Publications Center.

The following scenarios illustrate how different cell configurations might be affected.

Mixed node scenario

In this scenario, a cell consists of a WebSphere Application Server, Version 6.1 deployment manager and four nodes. Two of the nodes are WebSphere Application Server, Version 6.1 and two are WebSphere Application Server, Version 6.0. If a WebSphere MQ connection factory is defined at cell level and has custom properties defined which exploit the new fields available in WebSphere MQ, Version 6, then the

connection factory is only bound into the WebSphere Application Server cells that are at Version 6.1 level. The WebSphere Application Server, Version 6.0 nodes do not know about the new WebSphere MQ properties and do not bind into the JNDI. The enhancements made to WebSphere Application Server, Version 6.1 allow validation of the properties to be deferred to the WebSphere MQ client jars.

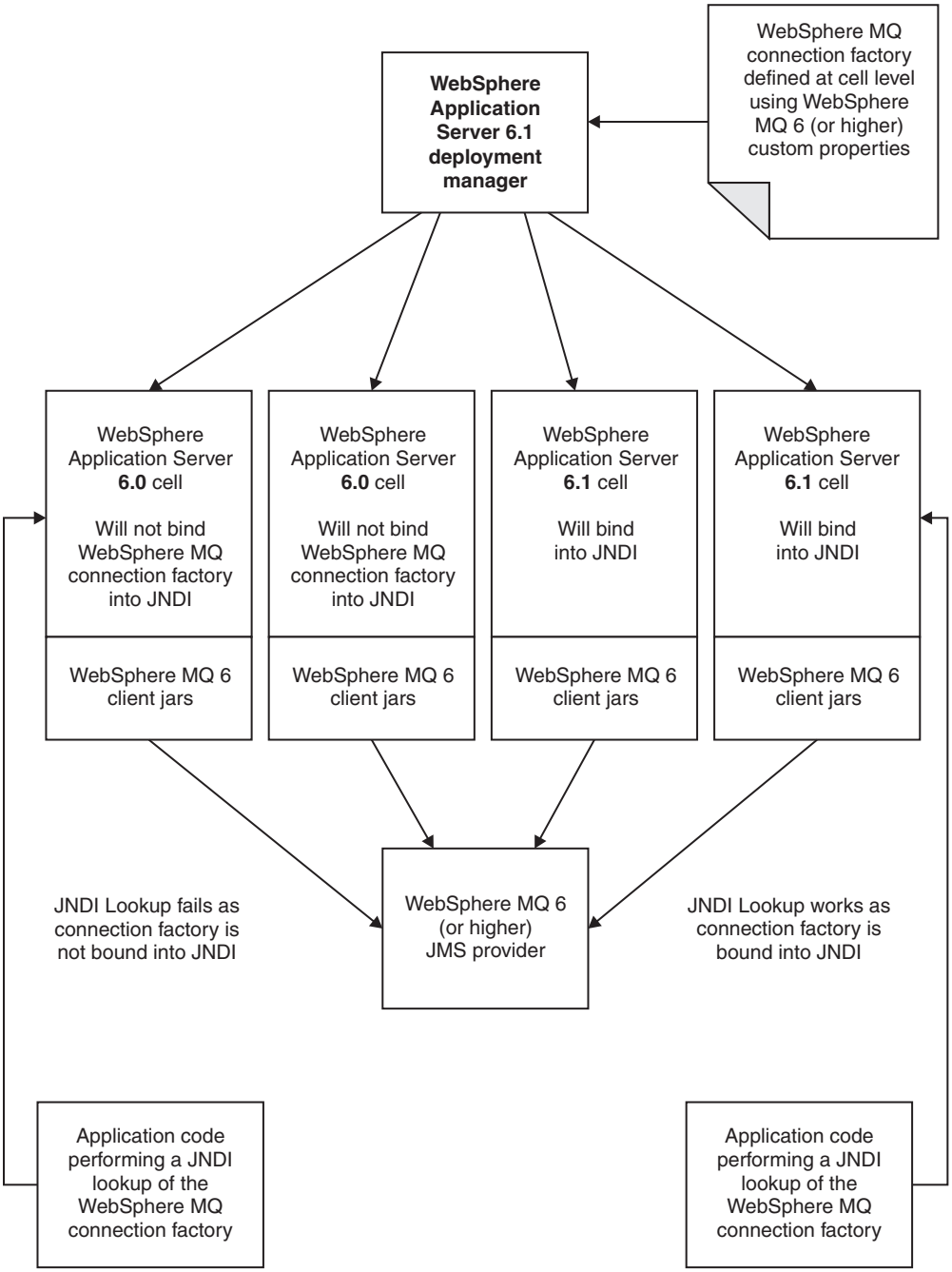


Figure 8. Mixed node scenario

Future version of WebSphere MQ scenario

In this scenario a cell consists of WebSphere Application Server, Version 6.1 deployment manager and nodes. The WebSphere MQ messaging provider is running at a level later than Version 6. WebSphere Application Server is using the default WebSphere MQ client jars shipped with WebSphere Application

Server, which are Version 6. In this scenario the WebSphere MQ client jars are not aware of the new WebSphere MQ properties so the validation fails and the connection factory does not bind into the JNDI.

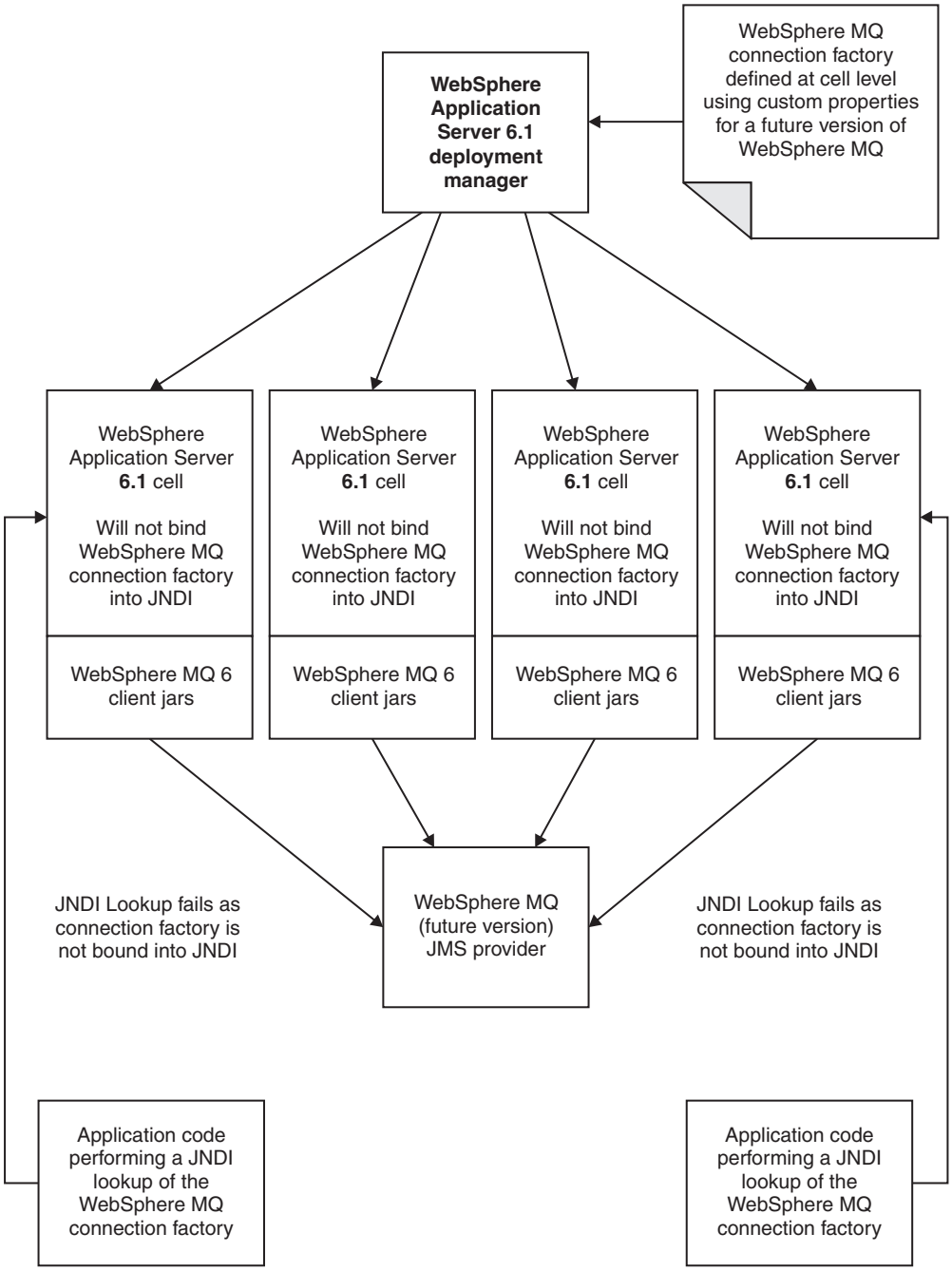


Figure 9. Future version of WebSphere MQ scenario

Correctly configured scenario

In this scenario, which is similar to the previous one, a cell consists of WebSphere Application Server, Version 6.1 deployment manager and nodes. The WebSphere MQ messaging provider is running at a level later than Version 6. To successfully use the new WebSphere MQ properties it is necessary to

configure the WebSphere Application Server to point to the WebSphere MQ client jars associated with the future version of WebSphere MQ.

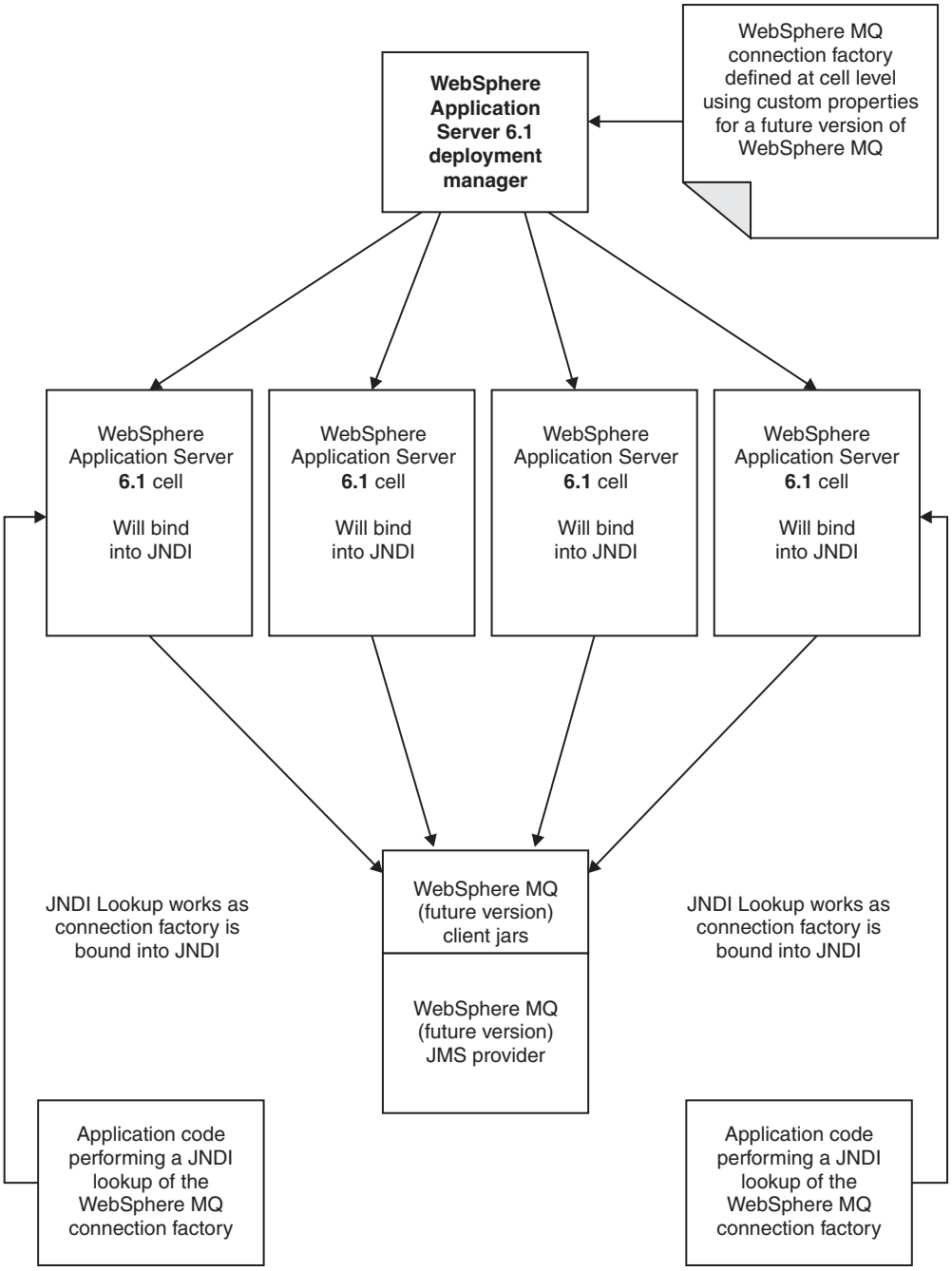


Figure 10. Correctly configured scenario

Error message example

The following example shows the type of text that the exception created by the client jars contains:

```
[09/02/06 15:40:06:377 GMT] 0000000a ContainerImpl E   WSVR0501E: Error creating
component null [class com.ibm.ws.runtime.component.ApplicationServerImpl]
com.ibm.ws.exception.RuntimeWarning: com.ibm.ws.runtime.component.binder.
ResourceBindingException: invalid configuration passed to resource binding logic.
REASON: Failed to create connection factory: Error raised constructing AdminObject,
error code: XAQCF PropertyName : XAQCF PropertyName
```

where `PropertyName` is the name of the invalid property.

Related tasks

"Configuring custom properties for the WebSphere MQ messaging provider" on page 961
You can use WebSphere MQ as a messaging provider for WebSphere Application Server. Many of the properties for a WebSphere MQ messaging provider can be selected from panels in the WebSphere Application Server administration console, but there are further properties that you can configure as custom properties. Use this task to create custom properties for destinations and connection factories when using WebSphere MQ as a messaging provider.

Securing WebSphere MQ messaging directories and log files

Use this task to restrict access to the /var/mqm directories and log files needed for WebSphere MQ as a JMS provider.

Note: The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations and prevent unauthorized access to the file system. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.

This procedure involves steps that you complete at different stages of installing and using IBM WebSphere Application Server, as described below. The steps are also described at appropriate points in other tasks, but are collected here for completeness.

1. Before installing WebSphere MQ, create and mount a file system called /var/mqm. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm.
2. Install WebSphere MQ as a messaging provider.

   As part of this stage, the installation program creates the /var/mqm/errors directory used to hold messaging logging files as well as the directories used to hold the messaging data. During the installation process these directories are secured with a default set of security attributes to prevent unauthoised access. If you change these permissions you should ensure that the permissions specified give WebSphere MQ messaging the required access.

Using JMS resources of a generic provider

This topic is the entry-point into a set of topics about enabling WebSphere applications to use JMS resources provided by a generic messaging provider (other than a WebSphere default messaging provider or WebSphere MQ).

You can install a messaging provider other than the default messaging provider or WebSphere MQ. WebSphere applications can use the JMS 1.1 interfaces or JMS 1.0.2 interfaces to access JMS resources provided by the generic messaging provider, in addition to JMS resources provided by the default messaging provider or WebSphere MQ (if installed).

You can use the WebSphere administrative console to administer the JMS connection factories and destinations provided by generic messaging providers.

In a mixed-version WebSphere Application Server deployment manager cell, you can administer generic messaging resources on both Version 6 and Version 5 nodes. For Version 5 nodes, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.

For more information about using a generic messaging provider to WebSphere Application Server, see the following topics:

- "Defining a generic messaging provider" on page 967

- “Listing generic JMS messaging resources”
- “Configuring JMS resources for a generic messaging provider” on page 973

Defining a generic messaging provider

Use this task to define a new messaging provider to WebSphere Application Server, for use instead of the default messaging provider or a WebSphere MQ as a messaging provider.

Before starting this task, you should have installed and configured the messaging provider and its resources by using the tools and information provided with the messaging provider.

To define a new generic messaging provider to WebSphere Application Server, use the administrative console to complete the following steps:

1. In the navigation pane, click **JMS Providers** → **Generic**. This displays the existing generic messaging providers in the content pane.
2. To define a new generic messaging provider, click **New** in the content pane. Otherwise, to change the definition of an existing messaging provider, click the name of the provider. This displays the properties used to define the messaging provider in the content pane.
3. Specify the following required properties. You can specify other properties, as described in a later step.

   Name The name by which this messaging provider is known for administrative purposes within IBM WebSphere Application Server.

   External initial context factory
   The Java classname of the initial context factory for the JMS provider.

   External provider URL
   The JMS provider URL for external JNDI lookups.
4. **Optional:** Click **Apply**. This enables you to specify additional properties.
5. **Optional:** Specify other properties for the messaging provider.

   Under Additional Properties, you can use the **Custom Properties** link to specify custom properties for your initial context factory, in the form of standard javax.naming properties.
6. Click **OK**.
7. Save the changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

You can now configure JMS resources for the generic messaging provider, as described in “Configuring JMS resources for a generic messaging provider” on page 973.

Listing generic JMS messaging resources

Use this task with the WebSphere administrative console to display administrative lists of JMS resources provided by a messaging provider other than the default messaging provider or WebSphere MQ.

You can use the WebSphere administrative console to display lists of the following types of JMS resources provided by a generic messaging provider. You can use the panels displayed to select JMS resources to administer, or to create or delete JMS resources (where appropriate).

To display administrative lists of JMS resources for a generic messaging provider, complete the following general steps:

1. Start the WebSphere administrative console.
2. In the navigation pane, click **Resources** → **JMS Providers** → **Generic**
3. If appropriate, in the content pane, change the scope of the generic messaging provider. If the scope is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.

4. In the content pane, under Additional Resources, click the link for the type of JMS resource. This displays a list of any existing resources of the selected type. For more information about the settings panels displayed for resources, see the related reference topics.

JMS provider collection

Use this panel to list JMS providers, or to select a JMS provider to view or change its configuration properties.

To view this administrative console page, click **Resources** → **JMS providers** → **Generic**

To view or change the properties of a JMS provider or its resources, select its name in the list displayed.

To define a new generic JMS provider, click **New**.

To act on one or more of the JMS providers listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Name The name by which this JMS provider is known for administrative purposes.

Description

A description of this JMS provider for administrative purposes.

For related information about JMS messaging providers and asynchronous messaging, see

- ../ae/cmj_jmsp.dita#cmj_jmsp.dita
- “Asynchronous messaging in WebSphere Application Server using JMS” on page 306

Generic JMS connection factory collection

The JMS connection factories configured in the associated generic messaging provider for both point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS connection factories, or to select a connection factory to browse or change its configuration properties.

This panel shows a list of the generic JMS connection factories with a summary of their configuration properties.

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

1. In the navigation pane, expand **Resources** → **JMS Providers** → **Generic**.
2. In the content pane, click the name of the generic messaging provider that you want to support the JMS connection factory.
3. Under Additional Properties, click **JMS connection factories**.

To define a new JMS connection factory, click **New**.

To view or change the properties of a JMS connection factory, select its name in the list displayed.

To act on one or more of the JMS connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Generic JMS connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS connection factory for use with the associated generic JMS provider. These configuration properties control how connections are created to the JMS destinations on the provider.

A JMS connection factory is used to create connections to JMS destinations. The JMS connection factory is created by the associated JMS provider.

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

1. In the navigation pane, expand **Resources** → **JMS Providers** → **Generic**.

2. In the content pane, click the name of the messaging provider that you want to support the JMS connection factory.
3. If appropriate, in the content pane, change the scope of the generic messaging provider.
4. Under Additional Properties, click **JMS connection factories**.
5. Click the name of the JMS connection factory that you want to work with.

A JMS connection factory for a generic JMS provider (other than the default messaging provider or WebSphere MQ as a JMS provider) has the following properties:

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the associated messaging provider.

Data type String

Type:

Whether this connection factory is for creating JMS queue destinations or JMS topic destinations.

Select one of the following options:

QUEUE
A JMS queue connection factory for point-to-point messaging.

TOPIC
A JMS topic connection factory for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the connection factory into the WebSphere Application Server name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

External JNDI name:

The JNDI name that is used to bind the connection factory into the name space of the generic messaging provider.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Mapping-Configuration Alias:

The module used to map authentication aliases.

This field provides a list of the modules that have been configured on the **Global Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum
Default Null
Range
ClientContainer
The client container maps authentication aliases.
WSLogin
The WSLogin module maps authentication aliases.
DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Custom properties:

An optional set of name and value pairs for custom properties passed to the messaging provider.

Generic JMS destination collection

The JMS destinations configured in the associated messaging provider for point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS destinations, or to select a JMS destination to browse or change its configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **Generic**.
2. In the content pane, click the name of the messaging provider that you want to support the JMS destination.
3. Under Additional Properties, click **JMS destinations**.

To define a new JMS destination, click **New**.

To view or change the properties of a JMS destination, select its name in the list displayed.

To act on one or more of the JMS destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Generic JMS destination settings:

Use this panel to browse or change the configuration properties of the selected JMS destination for use with the associated JMS provider.

A JMS destination is used to configure the properties of a JMS destination for the associated generic messaging provider (not the default messaging provider or WebSphere MQ). Connections to the JMS destination are created by the associated JMS connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS Providers** → **Generic**.
2. In the content pane, click the name of the messaging provider that you want to support the JMS destination.
3. Under Additional Properties, click **JMS destinations**.
4. Click the name of the JMS destination that you want to work with.

A JMS destination for use with a generic messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publish/subscribe).

Select one of the following options:
Queue
A JMS queue destination for point-to-point messaging.
Topic A JMS topic destination for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

External JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Configuring JMS resources for a generic messaging provider

Use the following tasks to configure the JMS connection factories and destinations for a generic messaging provider (not the default messaging provider or WebSphere MQ).

You only need to complete these tasks if your WebSphere Application Server environment uses a messaging provider other than the default messaging provider or WebSphere MQ to support enterprise applications that use JMS. To enable use of such a generic messaging provider, you must have installed and configured the messaging provider, as described in Defining a new JMS provider to WebSphere Application Server.

To configure JMS resources for a generic messaging provider, complete the following tasks:
- Configuring a JMS connection factory
- Configuring a JMS destination

Configuring a JMS connection factory for the generic JMS provider

Use this task to browse or change the properties of a JMS connection factory for use with a generic JMS provider (other than the default, V5 default or WebSphere MQ messaging providers).

To configure a JMS connection factory for use with a generic JMS provider, use the administrative console to complete the following steps:

1. Display the generic messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers** .
2. Select the default provider for which you want to configure a connection factory.
3. **Optional:** Change the **Scope** setting to the level at which the connection factory is visible to applications.
4. In the content pane, under Additional Properties, click **Connection factories** This displays a table listing any existing JMS connection factories, with a summary of their properties.
5. To browse or change an existing JMS connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server.

      Type Select whether the connection factory is for JMS queues (QUEUE) or JMS topics (TOPIC).

      JNDI Name
      The JNDI name that is used to bind the JMS connection factory into the WebSphere Application Server name space.

      External JNDI Name
      The JNDI name that is used to bind the JMS connection factory into the name space of the messaging provider.

   c. Click **Apply**. This defines the JMS connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
6. **Optional:** Change properties for the JMS connection factory, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS destination, a generic JMS provider

Use this task to browse or change the properties of a JMS destination for use with a generic JMS provider (other than the default messaging provider or WebSphere MQ).

To configure a JMS destination for use with a generic JMS provider, use the administrative console to complete the following steps:

1. Display the generic messaging provider. In the navigation pane, expand **Resources** → **JMS Providers** → **Generic**.
2. **Optional:** Change the **Scope** setting to the level at which the connection factory is visible to applications.
3. In the content pane, under Additional Properties, click **JMS destinations** This displays a table listing any existing JMS destinations, with a summary of their properties.
4. To browse or change an existing JMS destination, click its name in the list. Otherwise, to create a new destination, complete the following steps:
   a. Click **New** in the content pane.

b. Specify the following required properties. You can specify other properties, as described in a later step.

Name The name by which this JMS destination is known for administrative purposes within IBM WebSphere Application Server.

Type Select whether the destination is for JMS queues (QUEUE) or JMS topics (TOPIC).

JNDI Name
The JNDI name that is used to bind the JMS destination into the WebSphere Application Server name space.

External JNDI Name
The JNDI name that is used to bind the JMS destination into the name space of the messaging provider.

c. Click **Apply**. This defines the JMS destination to WebSphere Application Server, and enables you to browse or change additional properties.

5. **Optional:** Change properties for the JMS destination, according to your needs.
6. Click **OK**.
7. Save any changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

Administering support for message-driven beans

Use these tasks to manage resources used to support message-driven beans. These tasks are in addition to the tasks for administering resource adapters, JMS providers and the resources they provide.

You can use the WebSphere administrative console to configure the following resources for message-driven beans:

- J2C activation specifications for JCA 1.5-compliant message-driven beans. Activation specifications must be provided when the application's resources are configured using the default messaging provider or any generic J2C Resource Adapter that supports inbound messaging.
- The message listener service, listener ports, and listeners for EJB 2.0 message-driven beans deployed against listener ports. Listener ports must be provided when using the JMS providers: V5 Default Messaging, WebSphere MQ, or Generic.

You can update the configuration data at any time, but some updates only take effect when the appropriate server is next started.

For information about administering support for message-driven beans, see the following topics:
- Configuring JMS activation specifications, default messaging provider
- "Configuring a J2C activation specification"
- "Configuring a J2C administered object" on page 979
- Configuring message listener resources for EJB 2.0 message-driven beans

For other information about administering JMS providers and the messaging resources they provide, see the list of related topics.

Configuring a J2C activation specification

Use this task to configure a J2C activation specification used to deploy message-driven beans with an external resource adapter.

Use this task if you want to use a message-driven bean as a listener on a Java Connector Architecture (JCA) 1.5 resource adapter other than the default messaging JMS provider.

You can create or modify a J2C activation specification under an installed resource adapter at the cell, node, or server scope. You can select the message listener type from those provided by the given resource adapter.

Configuring a J2C activation specification offers two distinct advantages:

- The activation specification configuration information can be shared among multiple message-driven beans across multiple applications.
- Updates to the configuration properties can be made without the need to redeploy the application.

To configure a J2C activation specification for an external resource adapter, use the administrative console to complete the following steps. This task contains an optional step for you to create a new activation specification.

1. Display the external resource adapter. In the navigation pane, click **Resources** → **Resource Adapters** → *adapter_name*. This displays in the content pane a table of properties for the external resource adapter, including links to the types of J2C resources that it provides.
2. **Optional:** Change the **Scope** setting to the scope level at which the activation specification is to be visible to applications, according to your needs.
3. In the content pane, under the Activation specifications heading, click **J2C Activation Specifications**. This lists any existing J2C activation specifications for the external resource adapter in the content pane.
4. Display the properties of the J2C activation specification. If you want to display an existing J2C activation specification, click one of the names listed.

   Alternatively, if you want to create a new J2C activation specification, click **New**, then specify the following required properties:

   Name Type the name by which the activation specification is known for administrative purposes. The JNDI name is automatically generated based on the value for the Name property.

   Message listener type
   Select the message listener type that this activation specification instance should support. This list is based on the deployment descriptor of the external resource adapter.

   Depending on the external resource adapter, there can be additional required properties that need to be supplied. To provide values for these properties, click **Custom properties**. When creating a new activation specification, you may need to click **Apply** before this custom property selection is available.
5. Specify properties for the activation specification, according to your needs .
6. Click **OK**.
7. Save your changes to the master configuration.

J2C Activation Specifications collection

This page contains a list of J2C activation specifications for a resource adapter configuration and is used to create new J2C activation specifications, to select J2C activation specifications for configuration changes, or to delete J2C activation specifications.

Activation specification definitions and classes are provided by a resource adapter when it is installed. Using this information, the administrator can create and configure J2C activation specifications with JNDI names that are then available for applications to use. The resource adapter uses a J2C activation specification to configure a specific endpoint instance. Each application configuring one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application must use the activation specification to provide the configuration properties related to the processing of the inbound messages.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource adapters** > **Resource adapters** > *resource_adapter* > **J2C Activation Specifications**.

- **Resources** > **Resource adapters** > **J2C Activation Specifications**.

Name:

Specifies the display name of the J2C activation specification instance.

A string with no spaces meant to be a meaningful text identifier for the J2C activation specification.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the J2C activation specification instance.

Data type String

Scope:

Specifies the scope of the resource adapter that supports this activation specification. Only applications that are installed within this scope can use this activation specification.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this activation specification.

Description:

A free-form text string to describe the J2C activation specification instance.

Data type String

Message Listener Type:

The Message Listener Type that is used by this activation specification.

The list of available classes is provided by the resource adapter.

Data type String

J2C Activation Specifications settings:

Use this page to specify the settings for a J2C activation specification.

The resource adapter uses a J2C activation specification to configure a specific endpoint instance. Each application configuring one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application must use the activation specification to provide the configuration properties related to the processing of the inbound messages.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource adapters** > **Resource adapters** > *resource_adapter* > **J2C Activation Specifications** > *activation_specification*.
- **Resources** > **Resource adapters** > **J2C Activation Specifications** > *activation_specification*.

Name:

Specifies the display name of the J2C activation specification instance.

A string with no spaces meant to be a meaningful text identifier for the J2C activation specification. Name is required

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the J2C activation specification instance.

The JNDI name is required. If you do not specify one, it is created from the Name field. If not specified, the JNDI name defaults to *eis/[name]*

Data type String

Scope:

Specifies the scope of the resource adapter that supports this activation specification. Only applications that are installed within this scope can use this activation specification.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this activation specification.

Description:

A free-form text string to describe the J2C activation specification instance.

Data type String

Authentication alias:

This optional field is used to bind the J2C activation specification to an authentication alias (configured through the security JAAS screens).

This alias is used to access a user name and password that are set on the configured J2C activation specification. This field is only meaningful if the J2C activation specification you are configuring has a UserName and Password field.

Data type Text

Message Listener Type:

The Message Listener Type used by this activation specification.

For new objects, the list of available classes is provided by the resource adapter in a drop-down list. After you create the activation specification, the field is a read only text field.

Data type Drop-down list or text

Destination JNDIName:

The destination JNDIName field only appears when a message of type javax.jms.Destination with name *Destination* is received.

Configuring a J2C administered object

Use this task to configure a J2C administered object used to configure objects with an external resource adapter.

To configure a J2C administered object for an external resource adapter, use the administrative console to complete the following steps. This task contains an optional step for you to create a new administered object.

1. Display the external resource adapter. In the navigation pane, click **Resources** → **Resource Adapters** → *adapter_name*. This displays in the content pane a table of properties for the external resource adapter, including links to the types of J2C resources that it provides.
2. **Optional:** Change the **Scope** setting to the scope level at which the activation specification is to be visible to applications, according to your needs.
3. In the content pane, under the Additional Properties heading, click **J2C Administered Objects**. This lists any existing J2C administered objects for the external resource adapter in the content pane.
4. Display the properties of the J2C administered object. If you want to display an existing J2C administered object, click one of the names listed.

   Alternatively, if you want to create a new J2C administered object, click **New**, then specify the following required properties:

   Name Type the name by which the J2C administered object is known for administrative purposes. The JNDI name is automatically generated based on the value for the Name property.

   Administered object class
   Select the administered object class that this instance should support. This list is based on the deployment descriptor of the external resource adapter.

   Depending on the external resource adapter, there can be additional required properties that need to be supplied. To provide values for these properties, click **Custom properties**. When creating a new administered object, you may need to click **Apply** before this custom property selection is available.
5. Specify properties for the administered object, according to your needs .
6. Click **OK**.
7. Save your changes to the master configuration.

J2C Administered Objects collection

Use this page to specify administered object settings for a Resource Adapter.

Administered object definitions and classes are provided by a resource adapter when you install it. Using this information, the administrator can create and configure J2C administered objects with JNDI names that are then available for applications to use. Some messaging styles may need applications to use special administered objects for sending and synchronously receiving messages (through connection objects using messaging style specific APIs). It is also possible that administered objects may be used to perform transformations on an asynchronously received message in a message provider-specific way. Administered objects can be accessed by a component by using either a resource environment reference or a message destination reference (preferred).

You can access this administrative console page in one of two ways:

- **Resources** > **Resource adapters** > **Resource adapters** > *resource_adapter* > **J2C Administered Objects**.
- **Resources** > **Resource adapters** > **J2C Administered Objects**

Name:

Specifies display name assigned to this administered object.

Data type String

JNDI Name:

Specifies the JNDI name of the administered object.

Data type String

Scope:

Specifies the scope of the resource adapter that supports this administered object. Only applications that are installed within this scope can use this object.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this administrative object.

Description:

Specifies a description for the administered object.

Data type String

Administered Object class:

Specifies the Administered Object class that is associated with this J2C Administered object. This class must be one that is provided by the resource adapter.

Data type String

J2C Administered Object settings:

Use this page to specify the settings for an administered object.

Administered object definitions and classes are provided by a resource adapter when you install it. Using this information, the administrator can create and configure J2C administered objects with JNDI names that are then available for applications to use. Some messaging styles may need applications to use special administered objects for sending and synchronously receiving messages (through connection objects using messaging style specific APIs). It is also possible that administered objects may be used to perform transformations on an asynchronously received message in a message provider-specific way. Administered objects can be accessed by a component by using either a resource environment reference or a message destination reference (preferred).

You can access this administrative console page in one of two ways:

- **Resources** > **Resource adapters** > **Resource adapters** > *resource_adapter* > **J2C Administered Objects** > *J2C_administered_object*.
- **Resources** > **Resource adapters** > **J2C Administered Objects** > *J2C_administered_object*.

Name:

Specifies the name of the J2C administered object instance.

A string with no spaces meant to be a meaningful text identifier for the administered object. This name is required.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name that this administered object is bound under.

The JNDI name is required. If you do not specify one, it is created from the Name field. If not specified, the JNDI name defaults to *eis/[name]*

Data type String

Scope:

Specifies the scope of the resource adapter that supports this administered object. Only applications that are installed within this scope can use this object.

Data type String

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this administrative object.

Data type String

Description:

Specifies a text description of the J2C administered object instance.

Data type String

Administered Object Class:

For new objects, the list of available classes is provided by the resource adapter in a drop-down list. You can only select classes from this list.

After you create the administered object, you cannot modify the Administered Object Class; it is read only.

Data type Drop-down list or Text

Configuring message listener resources for message-driven beans

Use the following tasks to configure resources needed by the message listener service to support message-driven beans for use with a JMS provider that does not have a JCA 1.5 resource adapter.

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, such as the default messaging provider that is part of WebSphere Application Server version 6. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA resources to use a J2C

activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server version 5).

If you want to deploy an enterprise application to use JMS message-driven beans with a JMS provider that does not have a JCA 1.5 resource adapter, refer to the following subtopics:

Configuring the message listener service

Use this task to configure the properties of the message listener service for an application server, to support message-driven beans deployed against listener ports.

If the JMS provider does not have a JCA 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server version 5).

If you want to deploy an enterprise application to use message-driven beans with listener ports, you can use this task to browse or change the configuration of the message listener service for an application server.

To configure the message listener service for an application server, use the administrative console to complete the following steps:

1. Display the listener service settings page:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
2. **Optional:** Browse or change the value of properties for the message-driven bean thread pool.
   a. Click **Thread Pool**
   b. Change the following properties, to suit your needs:

      Minimum size
      The minimum number of threads to allow in the pool.

      Maximum size
      The maximum number of threads to allow in the pool.

      Thread inactivity timeout
      The number of milliseconds of inactivity that should elapse before a thread is reclaimed. A value of 0 indicates not to wait and a negative value (less than 0) means to wait forever.

      Note: The administrative console does not allow you to set the inactivity timeout to a negative number. To do this you must modify the value directly in the config.xml file.

      Allow thread allocation beyond maximum thread size
      Select this check box to enable the number of threads to increase beyond the maximum size configured for the thread pool.

   c. Click **OK**.
3. **Optional:** Specify any of the following optional properties that you need, as **Custom properties** of the message listener service:

   NON.ASF.RECEIVE.TIMEOUT, MQJMS.POOLING.TIMEOUT, MQJMS.POOLING.THRESHOLD, MAX.RECOVERY.RETRIES, and RECOVERY.RETRY.INTERVAL.

   For more information about these custom properties, see Custom Properties.

   To browse or change the properties, complete the following steps:

   a. Click **Custom properties**
   b. For each custom property, specify a value to suit your needs.

If you have not specified a property before:

1) Click **New**.
2) Type the name of the property.
3) Type the value of the property.
4) Click **OK**.

4. Save your changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the Application Server.

Message listener service:

The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.

This panel displays links to the Additional Properties pages for Listener Ports, Thread Pool, and Custom Properties for the message listener service.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service**

Custom Properties:

An optional set of name and value pairs for custom properties of the message listener service.

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL

Message listener port collection:

The message listener ports configured in the administrative domain

This panel displays a list of the message listener ports configured in the administrative domain. Each listener port is used with a message-driven bean to automatically receive messages from an associated JMS destination. You can use this panel to add new listener ports or to change the properties of existing listener ports.

To view this administrative console panel, click **Servers** → **Application Servers** → *application_server* → **[Messaging] Message Listener Service** → **Listener Ports**

For more information about asynchronous messaging, see “Asynchronous messaging in WebSphere Application Server using JMS” on page 306.

Listener port settings:

A listener port is used to simplify administration of the association between a connection factory, destination, and deployed message-driven bean.

Use this panel to view or change the configuration properties of the selected listener port.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Listener Ports** → *listener_port*

Name:

The name by which the listener port is known for administrative purposes.

Data type String
Default Null

Initial state:

The state that you want the listener port to have when the application server is next restarted

Data type Enum
Units Not applicable
Default Started
Range
Started
When the application server is next started, the listener port is started automatically.
Stopped
When the application server is next started, the listener port is not started automatically. If message-driven beans are to use this listener port on the application server, the system administrator must start the port manually or select the Started value of this property then restart the application server.

Description:

A description of the listener port, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Connection factory JNDI name:

The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`.

Data type String
Default Null

Destination JNDI name:

The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.

You cannot use a temporary destination for late responses.

Data type String
Default Null

Maximum sessions:

Specifies the maximum number of concurrent sessions that a listener can have with the JMS server to process messages.

Each session corresponds to a separate listener thread and therefore controls the number of concurrently processed messages. Adjust this parameter when the server does not fully use the available capacity of the machine and if you do not need to process messages in a specific message order.

Data type	Integer
Units	Sessions
Default	1
Range	1 through 2147483647
Recommended	• If you want to process messages in a strict message order, set the value to 1, so only one thread is ever processing messages. • If you want to process multiple messages simultaneously (known as “message concurrency”), set this property to a value greater than 1. Keep this value as low as possible to prevent overloading client applications. A good starting point for a 100% JMS workload with short transaction times is 2 to 4 sessions per processor. If longer running transactions exist, you may need more sessions, which should be determined by experimentation.

Maximum retries:

The maximum number of times that the listener tries to deliver a message before the listener is stopped, in the range 0 through 2147483647.

The maximum number of times that the listener tries to deliver a message to a message-driven bean instance before the listener is stopped.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
Range	0 (no retries) through 2147483647

Maximum messages:

The maximum number of messages that the listener can process in one transaction.

If the queue is empty, the listener processes each message when it arrives. Each message is processed within a separate transaction.

For the WebSphere V5 default messaging provider or WebSphere MQ as the JMS provider, if messages start accumulating on the queue then the listener can start processing messages in batches. For generic JMS providers, this property value is passed to the JMS provider but the effect depends on the JMS provider.

Data type	Integer
Units	Number of messages
Default	1
Range	1 through 2147483647

Recommended	For the WebSphere default messaging providers or WebSphere MQ as the JMS provider, if you want to process multiple messages in a single transaction, then set this value to more than 1. If messages start accumulating on the queue, then a value greater than 1 enables multiple messages to be batch-processed into a single transaction, and eliminates much of the overhead of transactions on JMS messages. **CAUTION:** • If one message in the batch fails processing with an exception, the entire batch of messages is put back on the queue for processing. • Any resource lock held by any of the interactions for the individual messages are held for the duration of the entire batch. • Depending on the amount of processing that messages need, and if XA transactions are being used, setting a value greater than 1 can cause the transaction to time out. If an XA transaction does time out routinely because processing multiple messages exceeds the transaction timeout, reduce this property to 1 (to limit processing to one message per transaction) or increase your transaction timeout.

Message listener service custom properties:

Use this panel to view or change an optional set of name and value pairs for custom properties of the message listener service.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Custom Properties**

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL

NON.ASF.RECEIVE.TIMEOUT:

The timeout in milliseconds for synchronous message receives performed by message-driven bean listener sessions in the non-ASF mode of operation.

You should set this property to a non-zero value only if you want to enable the non-ASF mode of operation for all message-driven bean listeners on the application server.

The message listener service has two modes of operation, Application Server Facilities (ASF) and non-Application Server Facilities (non-ASF).

- The ASF mode is meant to provide concurrency and transactional support for applications. For publish/subscribe message-drive beans, the ASF mode provides better throughput and concurrency, because in the non-ASF mode the listener is single-threaded.

- The non-ASF mode is mainly for use with generic JMS providers that do not support JMS ASF, which is an optional extension to the JMS specification. The non-ASF mode is also transactional but, because the path length is shorter than the ASF mode, usually provides improved performance.

  Use non-ASF if:
  - Your generic JMS provider does not provide JMS ASF support
  - You are using message-driven beans with WebSphere topic connections with the DIRECT port, because the embedded publish/subscribe broker using that port does not support XA transactions or JMS ASF.
  - Message order is a strict requirement

Data type Integer
Units Milliseconds
Default ASF mode (custom property not created)
Range 0 or greater milliseconds

0 non-ASF mode is disabled

1 or more
The timeout in milliseconds for non-ASF message-driven bean listener synchronous session receives

Recommended If a transaction timeout occurs, the message must recycle causing extra work. If you want to use the non-ASF mode, set this property to lower than the transaction timeout, but leave spare at least the maximum duration of your message-driven bean's onMessage() method. For example, if your message-driven bean's onMessage() method typically takes a maximum of 10 seconds, and the transaction timeout is set to 120 seconds, you might set the NON.ASF.RECEIVE.TIMEOUT property to no more than 110000 (110000 milliseconds, that is 110 seconds).

MQJMS.POOLING.TIMEOUT:

The number of milliseconds after which a connection in the pool is destroyed if it has not been used.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes.

Data type Integer
Units Milliseconds
Default 5 minutes
Range

MQJMS.POOLING.THRESHOLD:

The maximum number of unused connections in the pool.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if there are more than ten unused connections in the pool.

Data type Integer
Units Number of connections
Default 10
Range

MAX.RECOVERY.RETRIES:

The maximum number of times that a listener port managed by this service tries to recover from a failure before giving up and stopping. When stopped the associated listener port is changed to the stop state. The interval between retry attempts is defined by the RECOVERY.RETRY.INTERVAL custom property.

A failure can be one of two things:

- An unexpected error has occurred when a listener port tries to get a message from the JMS provider.
- The connection between the application server and the JMS provider has been lost, usually due to a network error.

Data type	Integer
Units	Retry attempts
Default	5
Range	0 (no retries) through 2147483647

RECOVERY.RETRY.INTERVAL:

The time in seconds between retry attempts by a listener port to recover from a failure. The maximum number of retry attempts is defined by the MAX.RECOVERY.RETRIES custom property.

A failure can be one of two things:

- An unexpected error has occurred when a listener port tries to get a message from the JMS provider.
- The connection between the application server and the JMS provider has been lost, usually due to a network error.

Data type	Integer
Units	Seconds
Default	60
Range	1 through 2147483647

Creating a new listener port

Use this task to create a new listener port for the message listener service, so that message-driven beans can be associated with the port to retrieve messages.

Although you can continue to deploy an EJB 2.0 message-driven bean against a listener port (as in WebSphere Application Server version 5), you are recommended to deploy such beans as JCA 1.5-compliant resources and to upgrade them to be EJB 2.1 message-driven beans.

If you want to deploy an enterprise application to use EJB 2.0 message-driven beans with listener ports, use this task to create a new listener port for a message-driven bean to retrieve messages from.

To create a new listener port, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
2. Click **New**.
3. Specify the following required properties:

Name The name by which the listener port is known for administrative purposes.

Connection factory JNDI name
The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`

Destination JNDI name
The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.

4. **Optional:** Change other properties for the listener port, according to your needs.
5. Click **OK**.
6. Save your changes to the master configuration.
7. To have the changed configuration take effect, stop then restart the application server.

If enabled, the listener port is started automatically when a message-driven bean associated with that port is installed.

Configuring a listener port

Use this task to browse or change the properties of an existing listener port, used by message-driven beans associated with the port to retrieve messages.

Although you can continue to deploy an EJB 2.0 message-driven bean against a listener port (as in WebSphere Application Server version 5), you are recommended to deploy such beans as JCA 1.5-compliant resources and to upgrade them to be EJB 2.1 message-driven beans.

If you have deployed an enterprise application to use EJB 2.0 message-driven beans with listener ports, use this task to browse or change the configuration of a listener port that a message-driven bean retrieves messages from.

To configure the properties of a listener port, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
2. Click the name of the listener port that you want to work with. This displays the properties of the listener port in the content pane.
3. **Optional:** Change properties for the listener port, according to your needs.
4. Click **OK**.
5. Save any changes to the master configuration.
6. To have a changed configuration take effect, stop then restart the application server.

Deleting a listener port

Use this task to delete a listener port from the message listener service, to prevent message-driven beans associated with the port from retrieving messages.

To delete a listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. Under Communications, click **Messaging** → **Message Listener Service** This displays the Message Listener Service properties in the content pane.

4. In the content pane, click **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, select the check box for the listener port that you want to delete.
6. Click **Delete**. This action stops the port (needed to allow the port to be deleted) then deletes the port.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Administering listener ports

Use the following tasks to administer listener ports, which each define the association between a connection factory, a destination, and a message-driven bean.

You can use the WebSphere administrative console to administer listener ports, as described in the following tasks.

- Adding a new listener port

  Use this task to create a new listener port, to specify a new association between a connection factory, a destination, and a message-driven bean. This enables deployed message-driven beans associated with the port to retrieve messages from the destination.
- Configuring a listener port

  Use this task to browse or change the configuration properties of a listener port.
- Starting a listener port

  Use this task to start a listener port manually.
- Stopping a listener port

  Use this task to stop a listener port manually.

Note: If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. You do not normally need to start or stop a listener port manually.

Starting a listener port:

Use this task to start a listener port on an application server, to enable the listeners for message-driven beans associated with the port to retrieve messages.

A listener is active, that is able to receive messages from a destination, if the deployed message-driven bean, listener port, and message listener service are all started. Although you can start these components in any order, they must all be in a started state before the listener can retrieve messages.

If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. However, you can start a listener port manually, as described in this topic.

When a listener port is started, the listener manager tries to start the listeners for each message-driven bean associated with the port. If a message-driven bean is stopped, the port is started but the listener is not started, and remains stopped. If you start a message-driven bean, the related listener is started.

To start a listener port on an application server, use the administrative console to complete the following steps:

1. If you want the listener for a deployed message-driven bean to be able to receive messages at the port, check that the message-driven bean has been started.
2. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.

3. Select the check box for the listener port that you want to start.
4. Click **Start**.
5. Save your changes to the master configuration.

Stopping a listener port:

Use this task to stop a listener port on an application server, to prevent the listeners for message-driven beans associated with the port from retrieving messages.

When you stop a listener port as described in this topic, the listener manager stops the listeners for all message-driven beans associated with the port.

To stop a listener port on an application server, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
2. Select the check box for the listener port that you want to stop.
3. Click **Stop**.
4. Save your changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the application server.

Message-driven beans - listener port components

The WebSphere Application Server support for message-driven beans deployed against listener ports is based on JMS message listeners and the message listener service, and builds on the base support for JMS.

The main components of WebSphere Application Server support for message-driven beans are shown in the following figure and described after the figure:

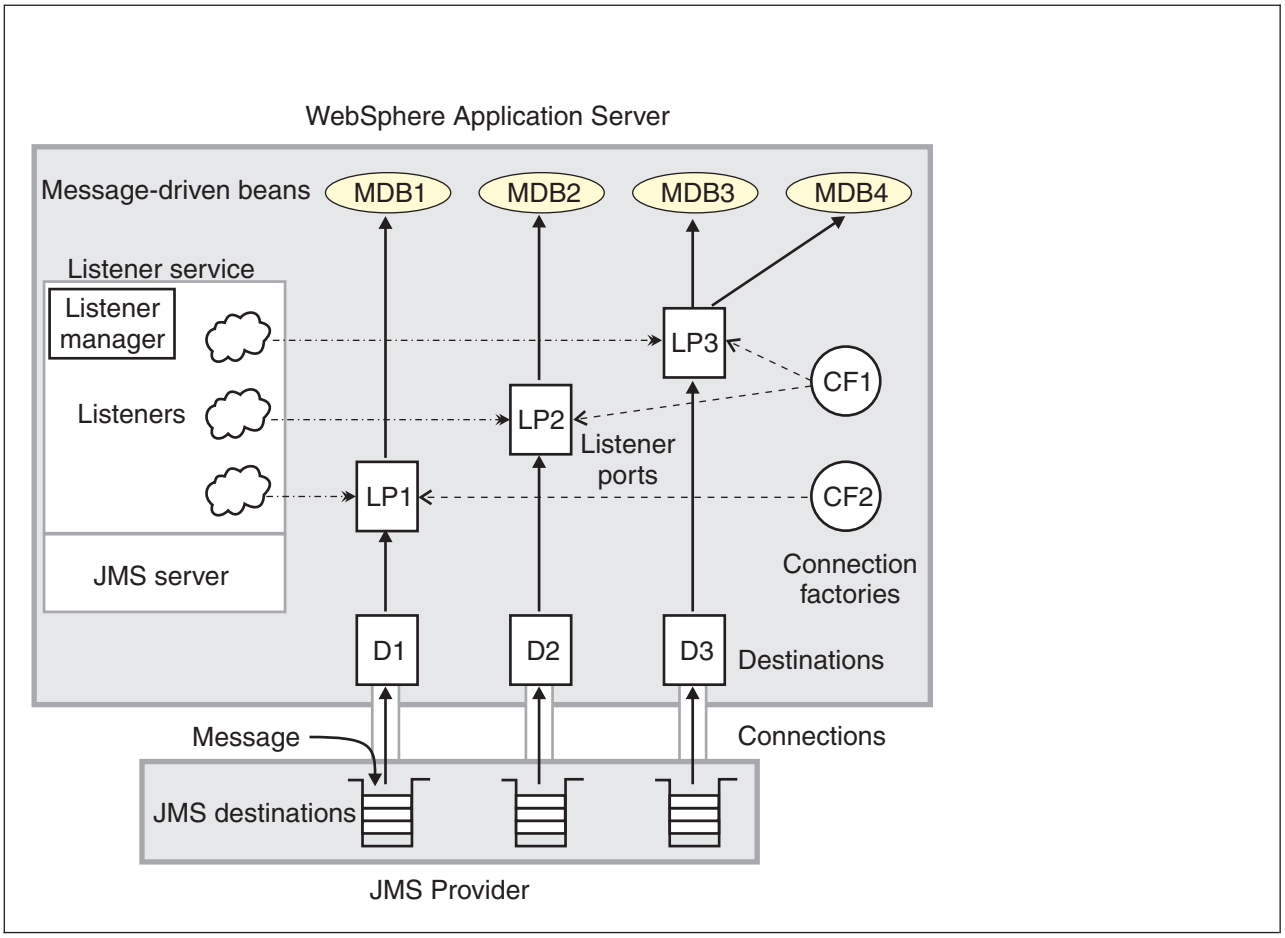


Figure 11. The main components for message-driven beans. This figure shows the main components of WebSphere support for message-driven beans, from JMS provider through a connection to a destination, listener port, then deployed message-driven bean that processes the message retrieved from the destination. Each listener port defines the association between a connection factory, destination, and a deployed message-driven bean. The other main components are the message listener service, which comprises a listener for each listener port, all controlled by the same listener manager. For more information, see the text that accompanies this figure.

The *message listener service* is an extension to the JMS functions of the JMS provider and provides a *listener manager*, which controls and monitors one or more JMS *listeners*.

Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for publish/subscribe messaging).

A *connection factory* is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A *listener port* defines the association between a connection factory, a destination, and a deployed *message-driven bean*. Listener ports are used to simplify the administration of the associations between these resources.

When a deployed message-driven bean is installed, it is associated with a listener port and the listener for a destination. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

When an application server is started, it initializes the listener manager based on the configuration data. The listener manager creates a dynamic session thread pool for use by listeners, creates and starts

listeners, and during server termination controls the cleanup of listener message service resources. Each listener completes several steps for the JMS destination that it is to monitor, including:

- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.
- Processing incoming messages by invoking the onMessage() method of the specified enterprise bean.

Important file for message-driven beans

The *server_name*-durableSubscriptions.ser file in the WAS_HOME/temp directory is important for the operation of the WebSphere Application Server messaging service, so should not be deleted.

If you do need to delete the WAS_HOME/temp directory or other files in it, ensure that you preserve the following file:

server_name-**durableSubscriptions.ser**

You should not delete this file, because the messaging service uses it to keep track of durable subscriptions for message-driven beans. If you uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription.

Chapter 15. Mail, URLs, and other J2EE resources

Using mail

Using the JavaMail API, a code segment can be embedded in any Java 2 Enterprise Edition (J2EE) application component, such as an EJB or a servlet, allowing the application to send a message and save a copy of the mail to the Sent folder.

The following is a code sample that you would embed in a J2EE application:

```
javax.naming.InitialContext ctx = new javax.naming.InitialContext();

   javax.mail.Session mail_session = (javax.mail.Session) ctx.lookup("java:comp/env/mail/MailSession3");
   MimeMessage msg = new MimeMessage(mail_session);

   msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse("bob@coldmail.net"));

   msg.setFrom(new InternetAddress("alice@mail.eedge.com"));

   msg.setSubject("Important message from eEdge.com");

   msg.setText(msg_text);

   Transport.send(msg);


   Store store = mail_session.getStore();

   store.connect();

   Folder f = store.getFolder("Sent");

   if (!f.exists()) f.create(Folder.HOLDS_MESSAGES);

   f.appendMessages(new Message[] {msg});
```

J2EE applications can use JavaMail APIs by looking up references to logically named mail connection factories through the `java:comp/env/mail` subcontext that is declared in the application deployment descriptor and mapped to installation specific mail session resources. As in the case of other J2EE resources, this can be done in order to eliminate the need for the application to hard code references to external resources.

1. Locate a resource through Java Naming and Directory Interface (JNDI). The J2EE specification considers a mail session instance as a resource, or a factory from which mail transport and store connections can be obtained. Do not hard code mail sessions (namely, fill up a Properties object, then use it to create a javax.mail.Session object). Instead, you must follow the J2EE programming model of configuring resources through the system facilities and then locating them through JNDI lookups.

   In the previous sample code, the line `javax.mail.Session mail_session = (javax.mail.Session) ctx.lookup("java:comp/env/mail/MailSession3");` is an example of not hard coding a mail session and using a resource name located through JNDI. You can consider the lookup name, `mail/MailSession3`, as a *soft link* to the real resource.
2. Define resource references while assembling your application. You must define a resource reference for the mail resource in the deployment descriptor of the component, because a mail session is referenced in the JNDI lookup. Typically, you can use an assembly tool shipped with WebSphere Application Server.

   When you create this reference, be sure that the name of the reference matches the name used in the code. For example, the previous code uses `java:comp/env/mail/MailSession3` in its lookup. Therefore

the name of this reference must be `mail/Session3`, and the type of the resource must be `javax.mail.Session`. After configuration, the deployment descriptor contains the following entry for the mail resource reference:

```
<resource-reference>
<description>description</description>
<res-ref-name>mail/MailSession3</res-ref-name>
<res-type>javax.mail.Session</res-type>
<res-auth>Container</res-auth>
```

3. Configure mail providers and sessions. The sample code references a mail resource, the deployment descriptor declares the reference, but the resource itself does not exist yet. Now you need to configure the mail resource that is referenced by your application component. Notice that the mail session you configure must have both its transport and mail access portions defined; the former required because the code is sending a message, the latter because it also saves a copy to the local mail store. When you configure the mail session, you need to specify a JNDI name. This is an important name for installing your application and linking up the resource references in your application with the real resources that you configure.
4. Install your application. You can install your application using either the administrative console or the scripting tool. During installation, WebSphere Application Server inspects all resource references and requires you to supply a JNDI name for each of them. This is not an arbitrary JNDI name, but the JNDI name given to a particular, configured resource that is the target of the reference.
5. Manage existing mail providers and sessions. You can update and remove mail providers and sessions.

   To update mail providers and sessions:

   a. Open the administrative console.
   b. Click **Resources** > **Mail** in the console navigation tree.
   c. Select the appropriate Java Mail resource to modify by clicking either **Mail Providers** or **Mail Sessions**.
   d. Select the specific resource to modify. To remove a mail provider or mail session, select the check box next to the appropriate resource and click **Delete**.
   e. Click **Apply** or **OK**.
   f. Save the configuration.
6. Enable debugger for a mail session.

If your application has a client, you can update mail providers and mail sessions using the Application Client Resource Configuration Tool (ACRCT).

JavaMail API

The JavaMail APIs provide a platform and protocol-independent framework for building Java-based mail client applications.

WebSphere Application Server supports the JavaMail API, Version 1.3, and the JavaBeans Activation Framework (JAF) Version 1.0. In WebSphere Application Server, the JavaMail API is supported in all Web application components, namely:

- Servlets
- JavaServer Pages (JSP) files
- Enterprise beans
- Application clients

The JavaMail APIs are generic for sending and reading mail. They require service providers, known in WebSphere Application Server as protocol providers, to interact with mail servers that run on pertaining protocols.

For example, Simple Mail Transfer Protocol (SMTP) is a popular transport protocol for sending mail. JavaMail applications can connect to an SMTP server and send mail through it by using this SMTP protocol provider.

In addition to service providers, the JavaMail API requires the Java Application Framework (JAF) to handle mail content that is not plain text, including Multipurpose Internet Mail Extensions (MIME), URL pages, and file attachments.

The JavaMail APIs, the JAF, the service providers, and the protocols are shipped as part of WebSphere Application Server. The API and related specifications are repackaged from Sun-licensed materials into the following file groupings:
- `j2ee.jar` - Contains the JavaMail API and the JAF
- `mail-impl.jar` - Contains the implementation of the JavaMail API
- `activation-impl.jar` - Contains the implementation of the JAF

Mail providers and mail sessions

A JavaMail service provider is a driver that supports JavaMail interaction with mail servers using a particular mail protocol. WebSphere Application Server includes service providers, also known as *protocol providers*, for mail protocols including Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP), and Post Office Protocol 3 (POP3).

A mail provider encapsulates a collection of protocol providers. For example, WebSphere Application Server has a built-in mail provider that encompasses the three protocol providers: SMTP, IMAP and POP3. These protocol providers are installed as the default and suffice for most applications.

If you have a particular application that requires custom protocol providers, you must first follow the steps outlined in the ″JavaMail API Design Specification, V1.2, Chapter 5 - The Mail Session″ to install your own protocol providers. This document outlines the process for the JavaMail 1.3 API as well as JavaMail 1.2. See the article, *Mail: Resources for learning*, for a link to the specification.

Mail sessions are represented by the `javax.mail.Session` class. A mail `Session` object authenticates users, and controls users' access to messaging systems.

To create platform-independent applications, use a resource factory reference to create a JavaMail session. A resource factory is an object that provides access to resources in the deployed environment of a program using the naming conventions defined by the Java Naming and Directory Interface (JNDI).

Ensure that every mail session is defined under a parent mail provider. Select a mail provider first and then create your new mail session.

JavaMail security permissions best practices

In many of its activities, the JavaMail API needs to access certain configuration files. The JavaMail and JavaBeans Activation Framework binary packages themselves already contain the necessary configuration files. However, the JavaMail API allows the user to define user-specific and installation-specific configuration files to meet special requirements.

The two locations where such configuration files can exist are `<user.home>` and `<java.home>/lib`. For example, if the JavaMail API needs to access a file named `mailcap` when sending a message, it first tries to access `<user.home>/.mailcap`. If that attempt fails, either due to lack of security permission or a nonexistent file, the API continues to try`<java.home> /lib/mailcap`. If that attempts also fails, it tries META-INF/mailcap in the class path, which actually leads to the configuration files contained in the `mail-impl.jar` and `activation-impl.jar` files. WebSphere Application Server uses the general-purpose JavaMail configuration files contained in the `mail-impl.jar` and `activation-impl.jar` files and does not put any mail configuration files in `<user.home>`and `<java.home>/lib`. File read permission for both the

mail-impl.jar and activation-impl.jar files is granted to all applications to ensure proper functioning of the JavaMail API, as shown in the following segment of the app.policy file:

```
grant codeBase "file:${application}" {
  // The following are required by Java mail
  permission java.io.FilePermission "${was.install.root}${/}lib${/}mail-impl.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}lib${/}activation-impl.jar", "read";
};
```

JavaMail code attempts to access configuration files at <user.home>and <java.home>/lib causing AccessControlExceptions to be thrown, since there is no file read permission granted for those two locations by default. This activity does not affect the proper functioning of the JavaMail API, but you might see a large amount of JavaMail-related security exceptions reported in the system log, which might swamp harmful errors that you are looking for. If this situation is a problem, consider adding two more permission lines to the permission block above. This should eliminate most, if not all, JavaMail-related harmless security exceptions from the log file. The application permission block in the app.policy file now looks like:

```
grant codeBase "file:${application}" {
  // The following are required by Java mail
  permission java.io.FilePermission "${was.install.root}${/}lib${/}mail-impl.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}lib${/}activation-impl.jar", "read";
java.io.FilePermission "${java.home}${/}lib${/}.mailcap", "read";
  permission java.io.FilePermission "${user.home}${/}lib${/}.mailcap", "read";
};
```

Mail: Resources for learning

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

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

Programming model and decisions

- JavaMail documentation

Programming specifications

- JavaMail 1.3 API documentation (Sun Java specifications)

JavaMail support for IPv6

WebSphere Application Server and its JavaMail component support Internet Protocol Version 6.0 (IPv6), meaning that:

- Both can run on a pure IPv4 network, a pure IPv6 network, *or* a mixed IPv4 and IPv6 network.
- On either the pure IPv6 network or the mixed network, the JavaMail component works with mail servers (such as the SMTP mail transfer agent, and the IMAP and POP3 mail stores) that are also IPv6 compatible. Additionally, a JavaMail component that is run on the mixed IPv4 and IPv6 network can communicate with mail servers using IPv4.

Use of brackets with IPv6 addresses

When you configure a mail session, you can specify the mail server hosts (also known as mail transport and mail store hosts) with domain-qualified host names or numerical IP addresses. Using host names is generally the preferred method. If you use IP addresses, however, consider enclosing IPv6 addresses in square brackets to prevent parsing inaccuracies. See the following example:

```
[fe80::202:57ff:fec4:2334]
```

The JavaMail API requires a combination of many host names or IP addresses with a port number, using the `host:port number` syntax . This extra colon can cause the port number to be read as part of an IPv6 address. Using brackets prevents your JavaMail implementation from processing the extra characters erroneously.

Using URL resources within an application

Java 2 Enterprise Edition (J2EE) applications can use Uniform Resource Locators (URLs) by looking up references to logically named URL connection factories through the `java:comp/env/url` subcontext, which is declared in the application deployment descriptor and mapped to installation specific URL resources.

As in the case of other J2EE resources, this can be done in order to eliminate the need for the application to hard code references to external resources. The process is the same used with other J2EE resources, such as JDBC objects and JavaMail sessions.

1. Develop an application that relies on naming features.
2. Define resource references while assembling your application. A URL resource that uses a built-in protocol, such as HTTP, FTP, or file, can use the default URL provider. URL resources that use other protocols need to use a custom URL provider.
3. Configure your URL resources within an application.
   a. Open the administrative console.
   b. Click **Resources**>**URL** in the console navigation tree.
   c. Click either **URL Providers** or **URLs** to modify the appropriate resource.
4. **Optional:** Configure URL providers and URLs within an application client using the Application Client Resource Configuration Tool (ACRCT).
5. Manage URL providers and URL resources used by the deployed application. To update or remove existing URL configurations:
   a. Open the administrative console.
   b. Click **Resources** > **URL** in the console navigation tree.
   c. Click either **URL Providers** or **URLs** to modify the appropriate resource.
   d. Select the URL to modify.
   e. Modify the URL properties.
   f. Click **Apply** or **OK**.

   To remove URL providers and URLs, after step 2, click *URL_provider* > **URLs**. Select the URL you want to remove and click **Delete**. Then, click **Apply** or **OK**.

URLs

A Uniform Resource Locator (URL) is an identifier that points to an electronically accessible resource, such as a directory file on a machine in a network, or a document stored in a database.

URLs appear in the format *scheme*:*scheme_information*.

You can represent a *scheme* as HTTP, FTP, file, or another term that identifies the type of resource and the mechanism by which you can access the resource.

In a World Wide Web browser location or address box, a URL for a file available using HyperText Transfer Protocol (HTTP) starts with `http:`. An example is `http://www.ibm.com`. Files available using File Transfer Protocol (FTP) start with `ftp:`. Files available locally start with `file:`.

The *scheme_information* commonly identifies the Internet machine making a resource available, the path to that resource, and the resource name. The *scheme_information* for HTTP, FTP and file generally starts with two slashes (//), then provides the Internet address separated from the resource path name with one slash (/). For example,

`http://www-4.ibm.com/software/webservers/appserv/library.html.`

For HTTP and FTP, the path name ends in a slash when the URL points to a directory. In such cases, the server generally returns the default index for the directory.

URL provider collection

Use this page to view existing URL providers, which supply the implementation classes that are necessary for WebSphere Application Server to access a URL through a specific protocol. The default URL provider provides connectivity through protocols that are supported by the IBM Developer Kit for the Java™ Platform, compatible with the Java 2 Standard Edition Platform 1.3.1. These protocols include HyperText Transfer Protocol (HTTP) and File Transfer Protocol (FTP), which work for must URLs.

To view this administrative console page, click **Resources** > **URL** > **URL Providers**.

Name

Specifies the administrative name for the URL provider.

Scope

Specifies the scope of this URL provider, which can support multiple URL configurations. All of the URL configurations that are supported by this provider inherit this scope.

Description

Describes the URL provider for your administrative records.

URL provider settings

Use this page to configure URL providers, which support WebSphere Application Server connections to a URL over a specific protocol.

To view this administrative console page, click **Resources** > **URL** > **URL Providers** > *URL_provider*.

Scope

Specifies the scope of this URL provider, which can support multiple URL configurations. All of the URL configurations that are supported by this provider inherit this scope.

Name

Specifies the administrative name for the URL provider.

Description

Describes the URL provider, for your administrative records.

Class path

Specifies paths or JAR file names which together form the location for the resource provider classes.

Stream handler class name

Specifies fully qualified name of a user-defined Java class that extends the `java.net.URLStreamHandler` class for a particular URL protocol, such as FTP.

Protocol

Specifies the protocol supported by this stream handler. For example, NNTP, SMTP, FTP.

URL collection

Use this page to view existing Uniform Resource Locator (URL) configurations, which are sets of properties that define WebSphere Application Server connections to URLs. URLs are location names that represent electronically accessible resources, such as a directory file on a machine in a network or a document stored in a database.

You can access this administrative console page in one of two ways:

- **Resources** > **URL Providers** > *URL_provider* > **URLs**
- **Resources** > **URL** > **URLs**

Name

Specifies the display name for the resource.

JNDI Name

Specifies the JNDI name.

Scope

Specifies the scope of the URL provider that supports this URL configuration. Only applications that are installed within this scope can use this URL configuration to access URL resources.

Provider

Specifies the URL provider that supplies the implementation classes for using a specific protocol to access this URL.

Description

Specifies the description of the resource.

Category

Specifies the category string, which you can use to classify or group the resource.

URL configuration settings

Use this page to define connections to Uniform Resource Locators (URLs), which are location names that represent electronically accessible resources. A collection of URL connection properties is often called a URL configuration in the WebSphere Application Server environment. The targeted resources are remote to your Application Server installation.

You can access this administrative console page in one of two ways:

- **Resources** > **URL** > **URLs** > *URL*
- **Resources** > **URL** > **URL Providers** > *URL_provider* > **URLs** > *URL*

Scope

Specifies the scope of the URL provider that supports this URL configuration. Only applications that are installed within this scope can use this URL configuration to access URL resources.

Provider

Specifies the URL provider that WebSphere Application Server uses for this URL configuration.

To create a new URL configuration: If you previously defined one or more URL providers at the relevant scope, you see a list from which you can select an existing URL provider for your new URL configuration.

Create New Provider

Provides the option of configuring a new URL provider for the new URL configuration.

Create New Provider is displayed only when you create a new URL from the **Resources** > **URL** > **URLs** path. In this flow, you can create a new URL provider if needed. The URL provider can not be changed during an edit.

Clicking **Create New Provider** triggers the console to display the URL provider configuration page, where you create a new provider. After you click **OK** to save your settings, you see the URL collection page. Click **New** to define a new URL configuration for use with the new provider; the console now displays a configuration page that lists the new provider as the URL configuration Provider.

Name
Specifies the display name for the resource.

JNDI Name
Specifies the JNDI name.

Description
Specifies the description of the resource.

Category
Specifies the category string, which you can use to classify or group the resource.

Specification
Specifies the string from which to form a URL.

URLs: Resources for learning
Use the following links to find relevant supplemental information about URLs. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

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

Programming specifications
- W3C Architecture - Naming and Addressing: URIs, URLs
- URL API documentation

Resource environment entries
This topic provides instructions on configuring *new* resource environment entries, which define environment resources that are the binding targets for resource-environment-references in an application's deployment descriptor.

1. Configure a resource environment provider, which is a library that provides the implementation for an environment resource factory. In the administration console, begin by clicking **Resources > Resource Environment > Resource Environment Providers > New**. (See the New Resource Environment Provider topic for more information.)
2. After saving your resource environment provider, go to the Additional Properties heading and click **Resource environment entries**. Click **New** to define a new resource environment entry. Refer to the "Resource environment entry settings" on page 1005 topic for descriptions of the required fields.
3. You also might need to create a referenceable, which specifies the factory class name that converts information in the name space into a class instance for your resource. To view the appropriate administrative console page for referenceables, click **Resources > Resource Environment > Resource Environment Providers >** *your_resource_environment_provider* > **Referenceables**. Click **New** to begin the configuration process. See the "Referenceables settings" on page 1006 topic for descriptions of the required fields.

Resource environment providers and resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object.

Not all objects bound into the server JNDI namespace are intended for use by an application client. For example, the WebSphere Application Server client run time does not support the use of Java 2 Connector (J2C) objects on the client. The object needs to be remotable, and the client-side implementations must be made available on the application client run-time classpath.

Resource environment references are different than resource references. Resource environment references allow your application client to use a logical name to look up a resource bound into the server JNDI namespace. A resource reference allows your application to use a logical name to look up a local J2EE resource. The J2EE specification does not specify a particular implementation of a resource.

Resource environment provider collection

Use this page to view resource environment providers, which encapsulate the referenceables that convert resource environment entry data into resource objects.

To view this administrative console page, click **Resources > Resource Environment > Resource Environment Providers**.

Name

Specifies a text identifier for the resource environment provider.

Data type String

Scope

Specifies the scope of this resource environment provider, which automatically becomes the scope of the resource environment entries that you define with this provider.

Description

Specifies a text string describing the resource environment provider.

Data type String

Resource environment provider settings

Use this page to create settings for a resource environment provider.

To view this administrative console page, click **Resources** > **Resource environment** > **Resource environment providers** > *resource environment provider*.

Scope:

Specifies the scope of this resource environment provider, which automatically becomes the scope of the resource environment entries that you define with this provider.

Name:

Specifies the name of the resource provider.

Data type String

Description:

Specifies a text description for the resource provider.

Data type String

New Resource environment provider

Use this page to define the configuration for a library that provides the implementation for a environment resource factory.

To view this administrative console page, click **Resources > Resource Environment > Resource Environment Providers > New**.

Scope:

Specifies the scope of this resource environment provider, which automatically becomes the scope of the resource environment entries that you define with this provider.

Name:

Specifies a text identifier for the resource environment provider.

Data type String

Description:

Specifies a text string describing the resource environment provider.

Data type String

Resource environment entries collection

Use this page to view configured resource environment entries. Within an application server name space, the data contained in a resource environment entry is converted into an object that represents a physical resource. This resource is frequently called an *environment resource*.

An environment resource can be of any arbitrary type. See the latest EJB specification for more information about resource environment references and environment resources.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource Environment** > **Resource environment entries**
- **Resources** >**Resource Environment** > **Resource Environment Providers** > *resource_environment_provider* > **Resource Environment Entries**

Name

Specifies a text identifier that helps distinguish this resource environment entry from others.

For example, you can use *My Resource* for the name.

Data type String

JNDI Name

Specifies the string to be used when looking up this environment resource using JNDI.

This is the string to which you bind resource environment reference deployment descriptors.

Data type String

Scope
Specifies the resource environment entry scope, which is inherited from the resource environment provider.

Provider
Specifies the resource environment provider for this entry. The provider encapsulates the classes that, when implemented, convert resource environment entry data into resource objects.

Description
Specifies text for information to help further identify and distinguish this resource

Data type String

Category
Specifies a category you can use to group environment resources according to some common feature.

It is strictly an organizational property and has no effect on the function of the environment resource.

Data type String

Resource environment entry settings
Use this page to configure resource environment entries. Within an application server name space, the data contained in a resource environment entry is converted into an object that represents a physical resource. Rather than represent a connection factory, which provides connections to a resource, this object *directly* represents a resource. This design can make the resource available to application modules that do not run entirely on the application server. Examples include some application clients and Web modules.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource Environment** > **Resource environment entries** > *resource_environment_entry*
- **Resources** >**Resource Environment** > **Resource Environment Providers** > *resource_environment_provider* > **Resource Environment Entries** > *resource_environment_entry*

Scope:

Specifies the scope of the resource environment provider, which is a library that supplies the implementation class for a resource environment factory. Within a JNDI name space, WebSphere Application Server uses the factory to transform your resource environment entry into an object that directly represents a physical resource.

Provider:

Specifies the resource environment provider.

Provider shows all of the existing resource environment providers that are defined at the relevant scope. Select one from the list if you want to use an existing resource environment provider as Provider.

Name:

Specifies a display name for the resource.

Data type String

JNDI name:

Specifies the JNDI name for the resource, including any naming subcontexts.

This name is used as the linkage between the platform's binding information for resources defined by a module's deployment descriptor and actual resources bound into JNDI by the platform.

Data type String

Description:

Specifies a text description for the resource.

Data type String

Category:

Specifies a category string that you can use to classify or group the resource.

Data type String

Referenceables:

Specifies the referenceable, which encapsulates the class name of the factory that converts resource environment entry data into a class instance for a physical resource.

Data type Drop-down menu

Referenceables collection

Use this page to view configured referenceables, which encapsulate the class name of the factory that converts information in the name space into a class instance for a physical resource.

To view this administrative console page, click **Resources** > **Resource environment** > **Resource Environment Providers** > *resource_environment_provider* > **Referenceables**.

Factory Class name

Specifies a javax.naming.spi.ObjectFactory implementation name

Data type String

Class name

Specifies the package name of the referenceable, for example: `javax.naming.Referenceable`

Data type String

Referenceables settings

Use this page to set the class name of the factory that converts information in the name space into a class instance of a physical resource.

To view this administrative console page, click **Resources > Resource Environment > Resource Environment Providers >** *resource_environment_provider* **> Referenceables >** *referenceable*.

Factory class name:

Specifies a javax.naming.ObjectFactory implementation class name

Data type String

Class name:

Specifies the Java type to which a Referenceable provides access, for binding validation and to create the reference.

Data type String

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** > **Enterprise Applications** > *application_name* > **Resource environment references**.

Guidelines for using this administrative console page:

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.

Data type String

Configuring mail providers and sessions

WebSphere Application Server includes a default mail provider called the *built-in* provider. If you use the default mail provider you only have to configure the mail session, which is the last step in this task. To use the customized mail provider you must first create the mail provider and session:

1. Open the administrative console.
2. Click **Resources** > **Mail**.
3. Click **Mail Providers**.
4. Create the mail provider.
   a. Select the scope for the new mail provider.
   b. Click **New**.
   c. Type the name of the mail provider in the name field.
   d. Click **Apply** or **OK**.
5. Define the protocol provider for the mail provider.
   a. Click *mail_provider*.
   b. Click **Protocol Providers**.
   c. Click **New**.
   d. Type the protocol name in the Protocol field.
   e. Type the class name in the Class name field.
   f. Click **Apply** or **OK**.

   Ensure that every mail session is defined under a parent mail provider. Select a mail provider first and then create your new mail session.
6. Create the mail session.
   a. Click *mail_provider*.
   b. Click **Mail Sessions**.
   c. Click **New**.
   d. Type the mail session name in the Name field.
   e. Type the JNDI name in the JNDI Name field.
   f. Click **Apply** or **OK**.
7. Configure the mail session.
   a. Click *mail_provider*.
   b. Click **Mail Sessions**.
   c. Click *mail_session*.
   d. Make changes to appropriate fields.
   e. Click **Apply** or **OK**.

If your application has a client you can configure JavaMail providers and sessions using the Application Client Resource Configuration Tool (ACRCT).

Mail provider collection

Use this page to view available JavaMail service providers, also known as *mail providers*. The mail provider encapsulates a collection of protocol providers, which implement the protocols for communication between your mail application and mail servers.

To view this administrative console page, click **Resources** > **Mail** > **Mail Providers**.

The **built-in mail provider** made available by WebSphere Application Server encompasses three protocol providers: Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP) and Post Office Protocol (POP3). Select the built-in provider if these protocols provide the right support for your mail system. If you have installed or plan to install different protocol providers, you must assign them a mail provider; select the scope at which you want the new mail provider to implement the protocols, then select **New**.

Name

Specifies the name of the JavaMail resource provider.

Scope

Specifies the scope in which the mail provider supports installed mail applications.

Description

Specifies the resource provider description.

Mail provider settings

Use this page to edit mail provider properties or configure a new mail provider (also called a *JavaMail service provider*). The mail provider encapsulates a collection of protocol providers, which implement the protocols for communication between your mail application and mail servers.

To view this administrative console page, click **Resources** > **Mail** > **Mail Providers** > *mail_provider*.

Scope

Specifies the scope in which the mail provider supports installed mail applications.

Name

Specifies the name of the JavaMail resource provider.

Description

Specifies the resource provider description.

Protocol providers collection

Use this page to select or add a protocol provider that supports interaction between your JavaMail application and mail servers. For example, your application might require the Simple Mail Transfer Protocol (SMTP), which is a popular transport protocol for sending mail. Selecting that protocol provider allows your JavaMail application to connect to an SMTP server, and send mail through the server.

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider* > **Protocol Providers**.

Protocol

Specifies the configuration of the protocol provider for a given protocol.

Class name
Specifies the implementation class for the specific protocol provider (also known as JavaMail service provider).

Class path
Specifies the path to the implementation class for the specific protocol provider (also known as JavaMail service provider).

Type
Specifies the type of protocol provider. Valid options are **STORE** or **TRANSPORT**.

Protocol providers settings
Use this page to set properties of a protocol provider, which provides the implementation class for a specific protocol to support communication between your JavaMail application and mail servers.

Built-in providers: WebSphere Application Server contains protocol providers for SMTP, IMAP and POP3. If you require custom providers for different protocols, install them in your application serving environment before configuring the providers. See the JavaMail API design specification for guidelines. After configuring your protocol providers, return to the mail provider page to find the link for configuring mail sessions.

To view this administrative console page, click **Resources** > **Mail** > **Mail Providers** > *mail_provider* > **Protocol Providers** > *protocol_provider*.

Scope
Specifies the scope at which this protocol provider was created. Only applications that are installed within this scope can use a protocol that is configured according to the settings of this protocol provider.

Protocol
Specifies the configuration of the protocol provider for a given protocol.

Class name
Specifies the implementation class of this protocol provider.

Class path
Specifies the path to the implementation class of this protocol provider.

Type
Specifies the type of protocol provider. Valid options are **STORE** or **TRANSPORT**.

Mail session collection
Use this page to view mail sessions that are defined under the parent mail provider.

You can access this administrative console page in one of two ways:

- **Resources** > **Mail** > **Mail Sessions**
- **Resources** > **Mail** > **Mail Providers** > *mail_provider* > **Mail Sessions**

Name
Specifies the administrative name of the JavaMail session object.

Scope
Specifies the scope at which the mail session was created. Only JavaMail applications that are installed in this scope can use this mail session.

Provider
Specifies the mail provider that WebSphere Application Server uses for this mail session.

JNDI Name
Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

This name provides the link between the platform binding information for resources defined in the client application deployment descriptor and the actual resources bound into JNDI by the platform.

Description
Specifies an optional description for your administrative records.

Category
Specifies an optional collection for classifying or grouping sessions.

Mail session settings
Use this page to configure mail sessions.

You can access this administrative console page in one of two ways:

- **Resources** > **Mail** > **Mail Sessions** > *mail_session*
- **Resources** > **Mail** > **Mail Providers** > *mail_provider* > **Mail Sessions** > *mail_session*

Scope
Specifies the scope of the mail provider that implements the JavaMail API for this mail session. Only applications that are installed within this scope can use this mail session.

Provider
Specifies the mail provider that WebSphere Application Server uses for this mail session.

When creating a mail session: If you previously defined one or more mail providers at the relevant scope, you see a list from which you can select an existing mail provider for your new mail session.

Create New Provider
Provides the option of configuring a new mail provider for the new mail session.

Create New Provider is displayed only when you click **Resources** > **Mail** > **Mail session** > **New** to create a new mail session.

Clicking **Create New Provider** triggers the console to display the mail provider configuration page, where you create a new provider. After you click **OK** to save your settings, you see the mail session collection page. Click **New** to define a new mail session for use with the new provider; the console now displays a configuration page that lists the new mail provider as the mail session Provider.

Remember: After you create a mail session, you cannot change the provider of that mail session.

Name
Specifies the administrative name of the JavaMail session object.

JNDI name
Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

This name provides the link between the platform binding information for resources that are defined in the client application deployment descriptor and the actual resources bound into JNDI by the platform.

Description
Specifies an optional description for your administrative records.

Category
Specifies an optional collection for classifying or grouping sessions.

Mail transport host
Specifies the server that is accessed when sending mail.

Mail transport protocol
Specifies the transport protocol that is used when sending mail.

Mail transport user ID
Specifies the user ID when the mail transport host requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Mail transport password
Specifies the password when the mail transport host requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Enable strict Internet address parsing
Specifies whether the recipient addresses must be parsed in strict compliance with RFC 822, which is a specifications document issued by the Internet Architecture Board.

This setting is not generally used for most mail applications. RFC 822 syntax for parsing addresses effectively enforces a strict definition of a valid e-mail address. If you select this setting, your JavaMail component adheres to RFC 822 syntax and rejects recipient addresses that do not parse into valid e-mail addresses (as defined by the specification). If you do not select this setting, your JavaMail component does not adhere to RFC 822 syntax and accepts recipient addresses that do not comply with the specification. By default, this setting is not selected. You can view the RFC 822 specification at the following Web address for the World Wide Web Consortium (W3C): http://www.w3.org/Protocols/rfc822/.

Mail from
Specifies the mail originator.

This value represents the Internet e-mail address that, by default, displays in the received message, as either the `From` or the `Reply-To` address. The recipient's reply comes to this address.

Mail store host
Specifies the server that is accessed when receiving the mail.

This setting, combined with the mail store user ID and password, represents a valid mail account. For example, if the mail account is *john_william@my.company.com*, then the mail store host is *my.company.com*.

Mail store protocol
Specifies the protocol that is used when receiving mail; it can be Internet Message Access Protocol (IMAP), Post Office Protocol 3 (POP3), or any store protocol for which an administrator has installed a provider.

Mail store user ID
Specifies the user ID for the given mail account.

For example, if the mail account is *john_william@my.company.com* then the user is *john_william*.

Mail store password
Specifies the password for the given mail account .

For example, if the mail account is *john_william@my.company.com* then enter the password for ID *john_william*.

Enable debug mode
Toggles debug mode on and off for this mail session.

Configuring mail, URLs, and resource environment entries with scripting

Use scripting to configure mail, URLs, and resource environment entries.

This topic contains the following tasks:

Configuring new mail providers using scripting

You can use scripting and the wsadmin tool to configure new mail providers.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new mail provider:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```
   - Using Jython:
     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:
   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```
2. Get required attributes:

- Using Jacl:

  ```
  $AdminConfig required MailProvider
  ```

- Using Jython:

  ```
  print AdminConfig.required('MailProvider')
  ```

  Example output:

  ```
  Attribute       Type
  name       String
  ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name MP1]
     set mpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'MP1']
     mpAttrs = [name]
     ```

4. Create the mail provider:
   - Using Jacl:

     ```
     set newmp [$AdminConfig create MailProvider $node $mpAttrs]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

     Example output:

     ```
     MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
     ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new mail sessions using scripting

You can use scripting and the wsadmin tool to configure new mail sessions.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new mail session:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

     Example output:

     ```
     MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
     ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required MailSession
     ```

   - Using Jython:

     ```
     print AdminConfig.required('MailSession')
     ```

     Example output:

```
Attribute       Type
name       String
jndiName       String
```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name MS1]
     set jndi [list jndiName mail/MS1]
     set msAttrs [list $name $jndi]
     ```

     Example output:

     ```
     {name MS1} {jndiName mail/MS1}
     ```

   - Using Jython:

     ```
     name = ['name', 'MS1']
     jndi = ['jndiName', 'mail/MS1']
     msAttrs = [name, jndi]
     print msAttrs
     ```

     Example output:

     ```
     [[name, MS1], [jndiName, mail/MS1]]
     ```

4. Create the mail session:
   - Using Jacl:

     ```
     $AdminConfig create MailSession $newmp $msAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('MailSession', newmp, msAttrs)
     ```

   Example output:

   ```
   MS1(cells/mycell/nodes/mynode|resources.xml#MailSession_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new protocols using scripting

You can configure new protocols with scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new protocol:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

   Example output:

   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required ProtocolProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('ProtocolProvider')
     ```

   Example output:

```
Attribute       Type
protocol      String
classname       String
```

3. Set up required attributes:
   - Using Jacl:

     ```
     set protocol [list protocol "Put the protocol here"]
     set classname [list classname "Put the class name here"]
     set ppAttrs [list $protocol $classname]
     ```

     Example output:

     ```
     {protocol protocol1} {classname classname1}
     ```

   - Using Jython:

     ```
     protocol = ['protocol', "Put the protocol here"]
     classname = ['classname', "Put the class name here"]
     ppAttrs = [protocol, classname]
     print ppAttrs
     ```

     Example output:

     ```
     [[protocol, protocol1], [classname, classname1]]
     ```

4. Create the protocol provider:
   - Using Jacl:

     ```
     $AdminConfig create ProtocolProvider $newmp $ppAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('ProtocolProvider', newmp, ppAttrs)
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#ProtocolProvider_4)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new custom properties using scripting

You can use scripting and the wsadmin tool to configure new custom properties.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new custom property:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

   Example output:

   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newmp propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newmp, 'propertySet')
     print propSet
     ```

Example output:

```
(cells/mycell/nodes/mynode|resources.xml#PropertySet_2)
```

3. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute       Type
   name      String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name CP1]
     set cpAttrs [list $name]
     ```

     Example output:

     ```
     {name CP1}
     ```

   - Using Jython:

     ```
     name = ['name', 'CP1']
     cpAttrs = [name]
     print cpAttrs
     ```

     Example output:

     ```
     [[name, CP1]]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $cpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, cpAttrs)
     ```

   Example output:

   ```
   CP1(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_2)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new resource environment providers using scripting

You can use the wsadmin tool and scripting to configure new resources environment providers.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new resource environment provider:

1. Identify the parent ID and assign it to the node variable.
   - Using Jacl:

     ```
     set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:
   - Using Jacl:
     ```
     $AdminConfig required ResourceEnvironmentProvider
     ```
   - Using Jython:
     ```
     print AdminConfig.required('ResourceEnvironmentProvider')
     ```

   Example output:
   ```
   Attribute     Type
   name     String
   ```
3. Set up the required attributes and assign it to the repAttrs variable:
   - Using Jacl:
     ```
     set n1 [list name REP1]
     set repAttrs [list $name]
     ```
   - Using Jython:
     ```
     n1 = ['name', 'REP1']
     repAttrs = [n1]
     ```
4. Create a new resource environment provider:
   - Using Jacl:
     ```
     set newrep [$AdminConfig create ResourceEnvironmentProvider $node $repAttrs]
     ```
   - Using Jython:
     ```
     newrep = AdminConfig.create('ResourceEnvironmentProvider', node, repAttrs)
     print newrep
     ```

   Example output:
   ```
   REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
   ```
5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring custom properties for resource environment providers using scripting

You can use scripting to configure custom properties for a resource environment provider.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new custom property for a resource environment provider:

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:
     ```
     set newrep [$AdminConfig getid /Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/]
     ```
   - Using Jython:
     ```
     newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/')
     print newrep
     ```

   Example output:
   ```
   REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
   ```
2. Identify the required attributes:
   - Using Jacl:
     ```
     $AdminConfig required J2EEResourceProperty
     ```
   - Using Jython:
     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

Example output:

```
Attribute     Type
name     String
```

3. Set up the required attributes and assign it to the repAttrs variable:
   - Using Jacl:

     ```
     set name [list name RP]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP']
     rpAttrs = [name]
     ```

4. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newrep propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newrep, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#PropertySet_1)
   ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new referenceables using scripting

You can use scripting and the wsadmin tool to configure new referenceables.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new referenceable:

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

     ```
     set newrep [$AdminConfig  getid  /Cell:mycell/Node:mynode/
     ResourceEnvironmentProvider:REP1/]
     ```

   - Using Jython:

     ```
     newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/
     ResourceEnvironmentProvider:REP1/')
     print newrep
     ```

   Example output:

   ```
   REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required Referenceable
     ```

- Using Jython:

```
print AdminConfig.required('Referenceable')
```

Example output:

```
Attribute     Type
factoryClassname   String
classname   String
```

3. Set up the required attributes:
   - Using Jacl:

```
set fcn [list factoryClassname REP1]
set cn [list classname NM1]
set refAttrs [list $fcn $cn]
```

   - Using Jython:

```
fcn = ['factoryClassname', 'REP1']
cn = ['classname', 'NM1']
refAttrs = [fcn, cn]
print refAttrs
```

   Example output:

```
{factoryClassname {REP1}} {classname {NM1}}
```

4. Create a new referenceable:
   - Using Jacl:

```
set newref [$AdminConfig create Referenceable  $newrep  $refAttrs]
```

   - Using Jython:

```
newref = AdminConfig.create('Referenceable',  newrep,  refAttrs)
print newref
```

   Example output:

```
(cells/mycell/nodes/mynode|resources.xml#Referenceable_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new resource environment entries using scripting

You can use scripting and the wsadmin tool to configure a new resource environment entry.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new resource environment entry:

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

```
set newrep [$AdminConfig  getid  /Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/]
```

   - Using Jython:

```
newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/')
print newrep
```

   Example output:

```
REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
```

2. Identify the required attributes:
   - Using Jacl:

```
$AdminConfig required ResourceEnvEntry
```

   - Using Jython:

```
print AdminConfig.required('ResourceEnvEntry')
```

Example output:

```
Attribute     Type
name      String
jndiName   String
referenceable   Referenceable@
```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name REE1]
     set jndiName [list jndiName myjndi]
     set newref [$AdminConfig getid /Cell:mycell/Node:mynode/Referenceable:/]
     set ref [list referenceable $newref]
     set reeAttrs [list $name $jndiName $ref]
     ```

   - Using Jython:

     ```
     name = ['name', 'REE1']
     jndiName = ['jndiName', 'myjndi']
     newref = AdminConfig.getid('/Cell:mycell/Node:mynode/Referenceable:/')
     ref = ['referenceable', newref]
     reeAttrs = [name, jndiName, ref]
     ```

4. Create the resource environment entry:
   - Using Jacl:

     ```
     $AdminConfig create ResourceEnvEntry $newrep $reeAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('ResourceEnvEntry', newrep, reeAttrs)
     ```

   Example output:

   ```
   REE1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvEntry_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring custom properties for resource environment entries using scripting

You can use scripting to configure a new custom property for a resource environment entry.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new custom property for a resource environment entry:

1. Identify the parent ID and assign it to the newree variable.
   - Using Jacl:

     ```
     set newree [$AdminConfig  getid  /Cell:mycell/Node:mynode/ResourceEnvEntry:REE1/]
     ```

   - Using Jython:

     ```
     newree = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvEntry:REE1/')
     print newree
     ```

   Example output:

   ```
   REE1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvEntry_1)
   ```

2. Create the J2EE custom property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newree propertySet]
     ```

   - Using Jython:

```
propSet = AdminConfig.showAttribute(newree, 'propertySet')
print propSet
```

Example output:

```
(cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_5)
```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name      String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP1]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP1']
     rpAttrs = [name]
     ```

5. Create the J2EE custom property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RPI(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new URL providers using scripting

You can use scripting and the wsadmin tool to configure new URL providers.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new URL provider:

1. Identify the parent ID and assign it to the node variable.
   - Using Jacl:

     ```
     set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required URLProvider
     ```

- Using Jython:

```
print AdminConfig.required('URLProvider')
```

Example output:

```
Attribute     Type
streamHandlerClassName   String
protocol       String
name         String
```

3. Set up the required attributes:
   - Using Jacl:

```
set name [list name URLP1]
set shcn [list streamHandlerClassName "Put the stream handler classname here"]
set protocol [list protocol "Put the protocol here"]
set urlpAttrs [list $name $shcn $protocol]
```

Example output:

```
{name URLP1} {streamHandlerClassName {Put the stream handler classname here}}
{protocol {Put the protocol here}}
```

   - Using Jython:

```
name = ['name', 'URLP1']
shcn = ['streamHandlerClassName', "Put the stream handler classname here"]
protocol = ['protocol', "Put the protocol here"]
urlpAttrs = [name, shcn, protocol]
print urlpAttrs
```

Example output:

```
[[name, URLP1], [streamHandlerClassName, "Put the stream handler classname here"],
[protocol, "Put the protocol here"]]
```

4. Create a URL provider:
   - Using Jacl:

```
$AdminConfig create URLProvider $node $urlpAttrs
```

   - Using Jython:

```
print AdminConfig.create('URLProvider', node, urlpAttrs)
```

Example output:

```
URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring custom properties for URL providers using scripting

You can use scripting to configure custom properties for URL providers.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure custom properties for URL providers:

1. Identify the parent ID and assign it to the newurlp variable.
   - Using Jacl:

```
set newurlp [$AdminConfig  getid  /Cell:mycell/Node:mynode/URLProvider:URLP1/]
```

   - Using Jython:

```
newurlp = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/')
print newurlp
```

Example output:

```
URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
```

2. Get the J2EE resource property set:
   - Using Jacl:
     ```
     set propSet [$AdminConfig showAttribute $newurlp propertySet]
     ```
   - Using Jython:
     ```
     propSet = AdminConfig.showAttribute(newurlp, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#PropertySet_7)
   ```
3. Identify the required attributes:
   - Using Jacl:
     ```
     $AdminConfig required J2EEResourceProperty
     ```
   - Using Jython:
     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name          String
   ```
4. Set up the required attributes:
   - Using Jacl:
     ```
     set name [list name RP2]
     set rpAttrs [list $name]
     ```
   - Using Jython:
     ```
     name = ['name', 'RP2']
     rpAttrs = [name]
     ```
5. Create a J2EE resource property:
   - Using Jacl:
     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP2(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```
6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring new URLs using scripting

You can use scripting and the wsadmin tool to configure new URLs.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following example to configure a new URL:

1. Identify the parent ID and assign it to the newurlp variable.
   - Using Jacl:
     ```
     set newurlp [$AdminConfig  getid  /Cell:mycell/Node:mynode/URLProvider:URLP1/]
     ```
   - Using Jython:
     ```
     newurlp = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/')
     print newurlp
     ```

   Example output:

```
URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
```

2. Identify the required attributes:
   - Using Jacl:
     ```
     $AdminConfig required URL
     ```
   - Using Jython:
     ```
     print AdminConfig.required('URL')
     ```

   Example output:
   ```
   Attribute     Type
   name          String
   spec          String
   ```
3. Set up the required attributes:
   - Using Jacl:
     ```
     set name [list name URL1]
     set spec [list spec "Put the spec here"]
     set urlAttrs [list $name $spec]
     ```

     Example output:
     ```
     {name URL1} {spec {Put the spec here}}
     ```
   - Using Jython:
     ```
     name = ['name', 'URL1']
     spec = ['spec', "Put the spec here"]
     urlAttrs = [name, spec]
     ```

     Example output:
     ```
     [[name, URL1], [spec, "Put the spec here"]]
     ```
4. Create a URL:
   - Using Jacl:
     ```
     $AdminConfig create URL $newurlp $urlAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('URL', newurlp, urlAttrs)
     ```

   Example output:
   ```
   URL1(cells/mycell/nodes/mynode|resources.xml#URL_1)
   ```
5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Configuring custom properties for URLs using scripting

Use the wsadmin tool and scripting to set custom properties for URLs.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to configure a new custom property for a URL:

1. Identify the parent ID and assign it to the newurl variable.
   - Using Jacl:
     ```
     set newurl [$AdminConfig getid /Cell:mycell/Node:mynode/URLProvider:URLP1/URL:URL1/]
     ```
   - Using Jython:
     ```
     newurl = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/URL:URL1/')
     print newurl
     ```

   Example output:
   ```
   URL1(cells/mycell/nodes/mynode|resources.xml#URL_1)
   ```

2. Create a J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newurl propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newurl, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_7)
   ```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP3]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP3']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP3(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_7)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
7. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Commands for the provider group of the AdminTask object

Use the commands in the provider group to create or delete a provider. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the provider group of the AdminTask object:

Table 8.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createProvider	The **createProvider** command creates a provider in the model given the ProviderInfo.	None	• Parameters: None • Returns: true if the provider was created, false if not.	**Interactive mode example usage:** • Using Jacl: `$AdminTask createProvider {-interactive}` • Using Jython string: `AdminTask.createProvider ('[-interactive]')` • Using Jython list: `AdminTask.createProvider (['-interactive'])`
deleteProvider	The **deleteProvider** command deletes a provider from the model given the providerName.	None	• Parameters: None • Returns: true if the provider was deleted, false if not.	**Interactive mode example usage:** • Using Jacl: `$AdminTask deleteProvider {-interactive}` • Using Jython string: `AdminTask.deleteProvider ('[-interactive]')` • Using Jython list: `AdminTask.deleteProvider (['-interactive'])`
getProviderInfo	The **getProviderInfo** command returns a ProviderInfo object defined in the model given the providerName.	None	• Parameters: None • Returns: A ProviderInfo object defined in the model given the providerName.	**Interactive mode example usage:** • Using Jacl: `$AdminTask getProviderInfo {-interactive}` • Using Jython string: `AdminTask.getProviderInfo ('[-interactive]')` • Using Jython list: `AdminTask.getProviderInfo (['-interactive'])`

Table 8. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getProviderInstance	The **getProviderInstance** command returns a java.security.Provider for the providerName specified.	None	• Parameters: None • Returns: A java.security.Provider for the providerName specified.	**Interactive mode example usage:** • Using Jacl: `$AdminTask getProviderInstance {-interactive}` • Using Jython string: `AdminTask.getProviderInstance ('[-interactive]')` • Using Jython list: `AdminTask.getProviderInstance (['-interactive'])`
getProviders	The **getProviders** command returns a List of ProviderInfo objects from the model.	None	• Parameters: None • Returns: A list of ProviderInfo objects from the model.	**Interactive mode example usage:** • Using Jacl: `$AdminTask getProviders {-interactive}` • Using Jython string: `AdminTask.getProviders ('[-interactive]')` • Using Jython list: `AdminTask.getProviders (['-interactive'])`

Chapter 16. Security

Task overview: Securing resources

WebSphere Application Server supports the Java 2 Platform, Enterprise Edition (J2EE) model for creating, assembling, securing, and deploying applications. Applications are often created, assembled, and deployed in different phases and by different teams.

You can secure resources in a J2EE environment by following the required high-level steps. Consult the J2EE specifications for complete details.

- Set up and enable security. You must address several issues prior to authenticating users, authorizing access to resources, securing applications, and securing communications. These security issues include migration, interoperability, and installation. After installing WebSphere Application Server, you must determine the proper level of security that is needed for your environment. For more information, see "Setting up and enabling security."
- Authenticate users. The process of authenticating users involves a user registry and an authentication mechanism. Optionally, you can define trust between WebSphere Application Server and a proxy server, configure single sign-on capability, and specify how to propagate security attributes between application servers. For more information, see "Authenticating users" on page 1082.
- Authorize access to resources. WebSphere Application Server provides many different methods for authorizing accessing resources. For example, you can assign roles to users and configure a built-in or external authorization provider. For more information, see "Authorizing access to resources" on page 1333.
- Secure communications. WebSphere Application Server provides several methods to secure communication between a server and a client. For more information, see "Securing communications" on page 1394.
- Develop extensions to the WebSphere security infrastructure. WebSphere Application Server provides various plug points so that you can extend the security infrastructure. For more information, see Developing extensions to the WebSphere security infrastructure.
- Secure various types of WebSphere applications. See **Securing WebSphere applications** for tasks involving developing, deploying, and administering secure applications, including Web applications, Web services, and many other types. This section highlights the security concerns and tasks that are specific to each type of application.
- Tune, harden, and maintain security configurations. After you have installed WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration. For more information, see Tuning, hardening, and maintaining.
- Troubleshoot security configurations. For more information, see Troubleshooting security configurations.

Your applications and production environment are secured.

See Security: Resources for learning for more information on the WebSphere Application Server security architecture.

Setting up and enabling security

You must address several issues prior to authenticating users, authorizing access to resources, securing applications, and securing communications. These security issues include migration, interoperability, and installation.

After installing WebSphere Application Server, you can determine the proper level of security that is needed for your environment. By default, administrative security is enabled and provides the authentication of users, the use of Secure Sockets Layer (SSL), and the choice of user account repository.

The following information is covered in this section:

- Determine if any migration and interoperability issues might affect your installation. For more information, see “Migrating, coexisting, and interoperating – Security considerations.”
- Prepare your environment before and after installing WebSphere Application Server. For more information, see “Preparing for security at installation time” on page 1045.
- Enable security for all your application servers or for specific application servers in your realm.

  For more information, see “Enabling security” on page 1047.

After installing WebSphere Application Server and securing your environment, you must authenticate users. For more information, see “Authenticating users” on page 1082.

Migrating, coexisting, and interoperating – Security considerations

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

This information addresses the need to migrate your security configurations from a previous release of IBM WebSphere Application Server to WebSphere Application Server Version 6.1 or later. Complete the following steps to migrate your security configurations:

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

Use the First steps wizard to access and run the Migration wizard.

1. Start the First steps wizard by launching the `firststeps.bat` or the `firststeps.sh` file. The first steps file is located in the following directory:
   - Linux *./app_server_root*`/profiles/`*profile_name*`/firststeps/firststeps.sh`
   - Windows *app_server_root*`\profiles\`*profile_name*`\firststeps\firststeps.bat`
2. On the First steps wizard panel, click **Migration wizard**.
3. Follow the instructions provided in the First steps wizard to complete the migration.

For more information on the Migration wizard, see Using the migration wizard to migrate product configurations.

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 6.1.

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

If you upgrade from WebSphere Application Server, Version 5.*x* to WebSphere Application Server, Version 6.1, the data that is associated with Version 5.x trust associations is not automatically migrated to Version 6.1. To migrate trust associations, see “Migrating trust association interceptors” on page 1036.

Interoperating with previous product versions

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

1. Configure WebSphere Application Server Version 6.1 with the same distributed user registry (that is, LDAP or Custom) that is configured with the previous version. Make sure that the same LDAP user registry is shared by all of the product versions.

a. In the administrative console, select **Security > Secure administration, applications, and infrastructure**.
b. Choose an available Realm definition and click **Configure**.
c. Enter a **Primary administrative user name**. This is the identity you use to login to the administrative console or to wsadmin. When inter-operating with a previous release, you must use the same server ID. In WebSphere Application Server Version 6.1, there are two choices for server ID. For previous releases, specify a server ID and password. However, if you choose to use the internal server ID, you must be able to override the generated value and to specify the server ID from a previous release.
d. Click either **Automatically generated server identity** or **Server identity that is stored in the user repository** .
e. If you select **Automatically generated server identity**, click **Authentication mechanisms and expiration** to change the identity to the one used by the previous release you are inter-operating with. Scroll down to the Cross-cell single sign-on section and enter the identity in **Internal server ID**.
f. If you select **Server identity that is stored in the user repository**, enter the **Server user id** and the associated **Password**.
g. Fill out the rest of the user registry settings and then click **OK**.

2. Configure the LTPA authentication mechanism. Automatic generation of the LTPA keys should be disabled. If not, keys used by a previous release are lost. Export the current LTPA keys from WebSphere Application Server Version 6.1 and import them into the previous release.
   a. In the administrative console select **Security > Secure administration, applications, and infrastructure**.
   b. Click **Authentication mechanisms and expiration**.
   c. Click the **Key set groups** link, then click the key set group that displays in the Key set groups panel.
   d. Clear the **Automatically generate keys** check box.
   e. Click **OK**, then click **Authentication mechanisms and expiration** in the path at the top of the Key set groups panel.
   f. Scroll down to the Cross-cell single sign-on section, and enter a password to use for encrypting the LTPA keys when adding them to the file.
   g. Enter the password again to confirm the password.
   h. Enter the **Fully qualified key file name** that contains the exported keys.
   i. Click **Export keys**.
   j. Follow the instructions provided in the previous release to import the exported LTPA keys into that configuration.
3. If you are using the default SSL configuration, extract all of the signer certificates from the WebSphere Application Server Version 6.1 common trust store. Otherwise, extract signers where necessary to import them into the previous release.
   a. In the administrative console, click **Security > SSL certificate and key management**.
   b. Click **Key stores and certificates**.
   c. Click **NodeDefaultTrustStore**.
   d. Click **Signer certificates**.
   e. Select one signer and click **Extract**.
   f. Enter a unique path and filename for the signer (for example, `c:\temp\signer1.arm`).
   g. Click **OK**. Repeat for all of the signers in the trust store.
   h. Check other trust stores for other signers that might need to be shared with the other server. Repeat steps e through h to extract the other signers.

4. Add the exported signers to DummyServerTrustFile.jks and DummyClientTrustFile.jks in the /etc directory of the back-level product version. If the previous release is not using the dummy certificate, the signer certificate(s) from the previous release must be extracted and added into the WebSphere Application Server Version 6.1 release to enable SSL connectivity in both directions.
   a. Open the key management utility, iKeyman, for that product version.
   b. Start ikeyman.bat or ikeyman.sh from the ${USER_INSTALL_ROOT}/bin directory.
   c. Select **Key Database File > Open**.
   d. Open ${USER_INSTALL_ROOT}/etc/DummyServerTrustFile.jks.
   e. Enter WebAS for the password.
   f. Select **Add** and enter one of the files extracted in step 2. Continue until you have added all of the signers.
   g. Repeat steps c through f for the DummyClientTrustFile.jks file.
5. Verify that the application uses the correct Java Naming and Directory Interface (JNDI) name and naming bootstrap port for performing a naming lookup.
6. Stop and restart all of the servers.

Interoperating with a C++ common object request broker architecture client

WebSphere Application Server supports security in the CORBA C++ client to access-protected enterprise beans. If configured, C++ CORBA clients can access protected enterprise bean methods using a client certificate to achieve mutual authentication on WebSphere Application Server applications.

You can achieve interoperability of Security Authentication Service between the C++ Common Object Request Broker Architecture (CORBA) client and WebSphere Application Server using Common Secure Interoperability Version 2 (CSIv2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP). The CSIv2 security service protocol has authentication, attribute and transport layers. Among the three layers, transport authentication is conceptually simple, however, cryptographically based transport authentication is the strongest. WebSphere Application Server has implemented the transport authentication layer, so that C++ secure CORBA clients can use it effectively in making CORBA clients and protected enterprise bean resources work together.

Security authentication from non-Java based C++ client to enterprise beans. WebSphere Application Server supports security in the CORBA C++ client to access-protected enterprise beans. If configured, C++ CORBA clients can access protected enterprise bean methods using a client certificate to achieve mutual authentication on WebSphere Application Server applications.

To support the C++ CORBA client in accessing protected enterprise beans:

- Create an environment file for the client, such as current.env. Set the variables presented in the following list in the file:

C++ security setting	Description
client_protocol_password	Specifies the password for the user ID.
client_protocol_user	Specifies the user ID to authenticate at the target server.
security_sslKeyring	Specifies the name of the RACF keyring for the client to use. The keyring must be defined under the user ID that is issuing the command to run the client.

- Point to the environment file using the fully qualified path name through the WAS_CONFIG_FILE environment variable. For example, in the test.sh test shell script, export:

```
/WebSphere/V6R0M0/DeploymentManager/profiles/default/config/cells
  /PLEX1Network/nodes/PLEX1Manager/servers/dmgr
```

  Some of the environment file terms are explained below:

default
profile name

PLEX1Network
cell name

PLEX1Manager
node name

dmgr server name

To support the C++ CORBA client in accessing protected enterprise beans:

1. Obtain a valid certificate to represent the client and export its public key to the target enterprise bean server.

   A valid certificate is needed to represent the C++ client. Request a certificate from the certificate authority (CA) or create a self-signed certificate for testing purposes.

   Use the Key Management Utility from the Global Security Kit (GSKit) to extract the public key from the personal certificate and save it in the `.arm` format.
2. Prepare a truststore file for WebSphere Application Server.

   Add the extracted client public key in the `.arm` file from the client to the server key truststore file. The server can now authenticate the client.

   Note: This is done by invoking the Key Management Utility through `ikeyman.bat` or `ikeyman.sh` from WebSphere Application Server installation.
3. Configure WebSphere Application Server to support Secure Sockets Layer (SSL) as the authentication mechanism.
   a. Start the administrative console.
   b. Locate the application server that has the target enterprise bean deployed and configure it to use SSL client certificate authentication.

      If it is a base installation, complete the following steps:

      1) Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 inbound authentication**. Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.
      2) Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 inbound transport** and verify that the **SSL-supported** option is selected.

      If it is a Network Deployment setting, complete the following steps:

      1) Click **Servers > Application Servers >** *server_name_where_the_EJB_resides*.
      2) Under Security, click **Server security**.
      3) Select the **RMI/IIOP security for this server overrides cell settings** option.
      4) Under Additional properties, click **CSIv2 inbound authentication**.
      5) Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.
      6) Click **Servers > Application Servers >** *server_name_where_the_EJB_resides*.
      7) Under Security, click **Server security**.
      8) Under Additional properties, click **CSIv2 inbound transport**.
      9) Verify that the **SSL-Supported** option is selected.
   c. Restart the application server.

      The WebSphere Application Server is ready to take a C++ CORBA security client and a mutually authenticated server and client by using SSL in the transport layer.
4. Configure the C++ CORBA client to use a certificate in performing the mutual authentication.

Client users are accustomed to using property files in their applications because they are helpful in specifying configuration settings. The following list presents important C++ security settings:

C++ security setting	Description
com.ibm.CORBA.bootstrapHostName=ricebella.austin.ibm.com	Specifies the target host name.
com.ibm.CORBA.securityEnabled=yes	Enables security.
com.ibm.CSI.performTLClientAuthenticationSupported=yes	Ensures client is supporting mutual authentication by certificate
com.ibm.ssl.keyFile=C:/ricebella/etc/DummyKeyRingFile.KDB	Specifies which key database file to use.
com.ibm.ssl.keyPassword=WebAS	Specifies the password for opening the key database file. WebSphere Application Server supports a utility called `PasswordEncode4cpp` to encode the plain password.
com.ibm.CORBA.translationEnabled=1	Enables the valueType conversion.

To use the property files in running a C++ client, an environment variable WASPROPS, is used to indicate where a property file or a list of property files exists.

For the complete set of C++ client properties, see the sample property file `scclient.props`, which is shipped with the product located in the *app_server_root*`/profiles/`*profile_name*`/etc` directory.

Migrating custom user registries

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

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

You cannot use other WebSphere Application Server components, for example, data sources, to initialize the custom registry because other components, like the containers, are initialized after security and are not available during the registry initialization. A custom registry implementation is a pure custom implementation, independent of other WebSphere Application Server components.

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

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

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

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

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

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

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

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

   ```
   public class FileRegistrySample implements CustomRegistry
   ```

   to:

   ```
   public class FileRegistrySample implements UserRegistry
   ```

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

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

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

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

   to:

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

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

4. Remove the getUsers method.

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

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

   to:

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

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

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

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

   to:

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

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

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

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

   to:

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

   In your implementation, construct the Result object from the list of the groups that is obtained from the user registry whose number is limited to the value of the limit parameter. Call the setHasMore method on the Result object if the total number of groups in the registry exceeds the limit value.
10. Add the createCredential method. This method is not called at this time, so return as `null`.

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

    The first and second lines of the previous code example are split onto two lines for illustrative purposes only.
11. To build the WebSphere Application Server Version 6.0.*x* and later implementation, make sure you have the `sas.jar` and the `wssec.jar` files in your class path.

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

    ```
    %install_root%/java/bin/javac -classpath %WAS_HOME%/lib/wssec.jar;
    %WAS_HOME%/lib/sas.jar FileRegistrySample.java
    ```

    Type the previous lines as one continuous line.

    To build the WebSphere Application Server Version 5 custom registry (CustomRegistry) in WebSphere Application Server Version 6.0.x and later, only the `sas.jar` file is required.
12. Copy the implementation classes to the product class path.

    The *%install_root%*`/lib/ext` directory is the preferred location.
13. Use the administrative console to set up the custom registry.

    Follow the instructions in “Configuring standalone custom registries” on page 1107 to set up the custom registry, including the **Ignore case for authorization** option. Make sure that you add the WAS_UseDisplayName properties if required.

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

If you are enabling security, see “Enabling security” on page 1047 to complete the remaining steps. When completed, save the configuration and restart all the servers. Try accessing some Java 2 Platform, Enterprise Edition (J2EE) resources to verify that the custom registry migration is successful.

Migrating trust association interceptors

Use this topic to manually migrate trust associations.

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

The following topics are addressed in this document:
- Changes to the product-provided trust association interceptors
- Migrating product-provided trust association interceptors
- Changes to the custom trust association interceptors
- Migrating custom trust association interceptors

Changes to the product-provided trust association interceptors

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

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

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

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

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

Migrating product-provided trust association interceptors

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

Changes to the custom trust association interceptors

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

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

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

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

Method 1:

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

Method 2:

Add the com.ibm.websphere.security.trustassociation.ignoreInitStatus property in the trust association interceptor custom properties. Set the value of this property to `true`, which tells

WebSphere Application Server to ignore the status of this method. If you add this property to the custom properties, WebSphere Application Server does not check the return status, which is similar to previous versions of WebSphere Application Server.

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

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

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

Method 1:

The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating `.config` to the com.ibm.websphere.security.trustassociation.types property value. If the `myTAI.properties` file is located in the *app_server_root*`/properties` directory, set the following properties:

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

Method 2:

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

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

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

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

Migrating custom trust association interceptors

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

1. Recompile the implementation file, if necessary.

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

   To recompile the implementation file, type the following code:

   ```
   %WAS_HOME%/java/bin/javac -classpath %WAS_HOME%/lib/wssec.jar;
   %WAS_HOME%/lib/j2ee.jar your_implementation_file.java
   ```

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

2. Copy the custom trust association interceptor class files to a location in your product class path. Copy these class files into the `%WAS_HOME%/lib/ext` directory.

3. Start WebSphere Application Server.
4. Enable security to use the trust association interceptor. The properties that are located in your custom trust association properties file and in the `trustedserver.properties` file are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.*x* and later using the trust association panels in the administrative console.

   For more information, see Configuring trust association interceptors.

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

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

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

The following list includes the deprecated CORBA programmatic login APIs.

- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/LoginHelper.java`.
- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java`.
- **org.omg.SecurityLevel2.Credentials**. This API is included with the product, but it is not recommended that you use the API.

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

The following information is only a summary; refer to the JAAS documentation for your platform located at: http://www.ibm.com/developerworks/java/jdk/security/ .

- Programmatic login APIs:
  - javax.security.auth.login.LoginContext
  - javax.security.auth.callback.CallbackHandler interface: The WebSphere Application Server product provides the following implementation of the javax.security.auth.callback.CallbackHandler interface:
    com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl
    Provides a non-prompt CallbackHandler handler when the application pushes basic authentication data (user ID, password, and security realm) or token data to product login modules. This API is recommended for server-side login.
    com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl
    Provides a login prompt CallbackHandler handler to gather basic authentication data (user ID, password, and security realm). This API is recommended for client-side login.

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

    WebSphere Application Server provides a LoginModules implementation for client and server-side login. Refer to the *Securing applications and their environment* PDF for details.

- javax.security.Subject:
  com.ibm.websphere.security.auth.WSSubject
  An extension provided by the product to invoke remote J2EE resources using the credentials in the javax.security.Subject
  com.ibm.websphere.security.cred.WSCredential
  After a successful JAAS login with the WebSphere Application Server LoginModules interfaces, a com.ibm.websphere.security.cred.WSCredential credential is created and stored in the Subject.
  com.ibm.websphere.security.auth.WSPrincipal
  An authenticated principal that is created and stored in a Subject that is authenticated by the WebSphere Application Server LoginModules interface.

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

   Note: The LoginHelper application programming interface (API) that is used in the following example is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. It is recommended that you use the JAAS programmatic login APIs that are shown in the next step.

```
public class TestClient {
...
private void performLogin() {
// Get the ID and password of the user.
String userid = customGetUserid();
String password = customGetPassword();

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

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

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

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

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

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

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

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by  login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
```

```
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

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

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

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

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

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

Migrating from the CustomLoginServlet class to servlet filters

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

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

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

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

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

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

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

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

Migrating Java 2 security policy

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

Previous WebSphere Application Server releases

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

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

The following two permissions are enforced by the Java 2 security manager (hard coded) for WebSphere Application Server:

- java.lang.RuntimePermission(exitVM)
- java.lang.RuntimePermission(setSecurityManager)

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

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

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

What changed

Java 2 Security is fully supported in WebSphere Application Server, which means that all permissions are enforced. The default Java 2 security policy for an enterprise application is the recommended permission set defined by the Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 specification. Refer to the *profile_root*/config/cells/*cell_name*/nodes/*node_name*/app.policy file for the default Java 2 security policy that is granted to enterprise applications. This policy is a much more stringent compared to previous releases.

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

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

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

Migrating system properties

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

used; otherwise ${was.install.root} is used. Use these properties interchangeably for the WebSphere Application Server or the Network Deployment environments. See the steps for migrating Java 2 security policy.

Migrating the Java 2 Security Policy

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

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

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

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

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

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

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

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

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

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

   You also can use an assembly tool to validate the contents of the `was.policy` file. For more information, see the *Securing applications and their environment* PDF.
4. Validate that the enterprise application does not require additional permissions to the migrated Java 2 security permissions and the default permissions set declared in the `${user.install.root}/config/cells/`*cell_name*`/nodes/`*node_name*`/app.policy` file. This validation requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a preproduction environment. Refer to developer kit APIs protected by

Java 2 security for information about which APIs are protected by Java 2 security. If you use third-party libraries, consult the vendor documentation for APIs that are protected by Java 2 security. Verify that the application is granted all the required permissions, or it might fail to run when Java 2 security is enabled.

5. Perform preproduction testing of the migrated enterprise application with Java 2 security enabled. Enable trace for the WebSphere Application Server Java 2 security manager in a preproduction testing environment with the following trace string: `com.ibm.ws.security.core.SecurityManager=all=enabled`. This trace function can be helpful in debugging the AccessControlException exception that is created when an application is not granted the required permission or some system code is not properly marked as privileged. The trace dumps the stack trace and permissions that are granted to the classes on the call stack when the exception is created.

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

   Note: Because the Java 2 security policy is much more stringent compared with previous releases, the administrator or deployer must review their enterprise applications to see if extra permissions are required before enabling Java 2 security. If the enterprise applications are not granted the required permissions, they fail to run.

Preparing for security at installation time

Complete the following tasks to implement security before, during, and after installing WebSphere Application Server.

1. Secure your environment before installation. This step describes how to perform WebSphere Application Server installation with proper authority on different platforms. For more information refer to "Securing your environment before installation."
2. Prepare the operating system for installation of WebSphere Application Server. This step describes how to prepare the different operating systems for installation of WebSphere Application Server. For more information, see Preparing the operating system for product installation.
3. Migrate security configurations from previous releases during installation, when you are prompted to do so. This step describes how to migrate security configurations from a previous release of WebSphere Application Server to WebSphere Application Server Version 6.1.

   For more information, see Migrating product configurations.
4. **Optional:** You can create a profile during install time. If you elect to do so, administrative security is enabled for that profile ″out of the box″ by default. A panel is displayed during profile creation time and **enabling administrative security** is selected by default. If you elect to keep this as the default, you must supply an administrative user ID and password. This user ID is created in a federated repository, which is the default user registry when enabling administrative security at profile creation time.
5. Secure your environment after installation. This step provides information on how to protect password information after you install WebSphere Application Server. For more information, see "Securing your environment after installation" on page 1046.

Securing your environment before installation

The following instructions explain how to perform a product installation with proper authority on UNIX platforms, Linux platforms, Solaris operating environments, and Windows platforms. These instructions apply when you plan to use the local operating system as your user registry.

Windows platforms:

On Windows platforms, the logon user must be a member of the administrator group with the rights of **Log on as a service**.

To add the rights to a user on a Windows platform:

1. Click **Start** > **Programs** > **Administrative Tools** > **Local Security Policy** (for domain configuration, select **Domain Security Policies**, instead).

2. From the Local Security Settings Panel, click **Local Policies** > **User Rights Assignment** and add the following rights to the user ID:
   - Log on as a service

Securing your environment after installation

WebSphere Application Server depends on several configuration files that are created during installation. These files contain password information and need protection. Although the files are protected to a limited degree during installation, this basic level of protection is probably not sufficient for your site. Verify that these files are protected in compliance with the policies of your site.

Note: A Kerberos keytab configuration file contains a list of keys that are analogous to user passwords. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk, which makes them readable only by authorized users.

The files in the *app_server_root*/profiles/*profile_name*/config and *app_server_root*/profiles/*profile_name*/properties , except for those in the following list, need protection. For example, give permission to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Other users or groups, such as WebSphere Application Server console users and console groups need permissions as well.

The files in the *app_server_root*/profiles/*profile_name*/properties directory that should not be protected are:

- TraceSettings.properties
- client.policy
- client_types.xml
- implfactory.properties
- sas.client.props
- sas.stdclient.properties
- sas.tools.properties
- soap.client.props
- wsadmin.properties
- wsjaas_client.conf
- sas.server.props
- server.policy
- ssl.client.props
- was.policy

Windows Secure files on a Windows system:

1. Open the browser for a view of the files and directories on the machine.
2. Locate and right-click the file or the directory that you want to protect.
3. Click **Properties**.
4. Click the **Security** tab.
5. Remove the Everyone entry and any other user or group that you do not want to have access to the file.
6. Add the users who can access the files with the proper permission.

After securing your environment, only the users with permission can access the files. Failure to adequately secure these files can lead to a breach of security in your WebSphere Application Server applications.

If failures occur that are caused by file accessing permissions, check the permission settings.

Enabling security

It is helpful to understand security from an infrastructure perspective so that you know the advantages of different authentication mechanisms, user registries, authentication protocols, and so on. Picking the right security components to meet your needs is a part of configuring security. The following sections help you make these decisions.

Read the following article before continuing with the security configuration:
- “Administrative security” on page 1050
- Security

After you understand the security components, you can proceed to configure security in WebSphere Application Server.

Note: For WebSphere Application Server Version 6.1, administrative security is enabled by default whenever a new profile is created, either during the initial install when you create a new profile or during post-install when you use the profile creation tooling. You can decide not to enable administrative security during profile creation time by instead enabling security post-profile creation using the administrative console.

1. Start the WebSphere Application Server administrative console.

   If security is currently disabled, you are prompted for a user ID. Log in with any user ID. However, if security is currently enabled, you are prompted for both a user ID and a password. Log in with a predefined administrative user ID and password.
2. Click **Security > Secure administration, applications, and infrastructure**. Use the Security Configuration Wizard, which is now available in WebSphere Application Server, Version 6.1, or configure security manually. The configuration order is not important. For more information on manual configuration, see tsec_authusers.dita.
3. Configure the user account repository. For more information, see “Selecting a registry or repository” on page 1083. On the Secure administration, applications, and infrastructure panel, you can configure user account repositories such as federated repositories, local operating system, standalone Lightweight Directory Access Protocol (LDAP) registry, and standalone custom registry.

   Note: You can choose to specify either a server ID and password for interoperability or enable a WebSphere Application Server 6.1 installation to automatically generate an internal server ID. For more information about automatically generating server IDs, see “Local operating system settings” on page 1087.

   One of the details common to all user registries or repositories is the *Primary administrative user name*. This ID is a member of the chosen repository, but also has special privileges in WebSphere Application Server. The privileges for this ID and the privileges that are associated with the administrative role ID are the same. The Primary administrative user name can access all of the protected administrative methods.

   Windows The ID must not be the same name as the machine name of your system because the repository sometimes returns machine-specific information when querying a user of the same name.

   In standalone LDAP registries, verify that the Primary administrative user name is a member of the repository and not just the LDAP administrative role ID. The entry must be searchable.

   The Primary administrative user name does **not** run WebSphere Application Server processes. Rather, the process ID runs the WebSphere Application Server processes.

   The *process ID* is determined by the way the process starts. For example, if you use a command line to start processes, the user ID that is logged into the system is the process ID. If running as a service, the user ID that is logged into the system is the user ID running the service. If you choose the local operating system registry, the process ID requires special privileges to call the operating system APIs. The process ID must have the following platform-specific privileges:

   - Windows **Act as Part of Operating System** privileges

4. Select the **Set as current** option after you configure the user account repository. When you click **Apply** and the Enable administrative security option is set, a verification occurs to see if an administrative user ID has been configured and is present in the active user registry. The administrative user ID can be specified at the active user registry panel or from the console users link. If you do not configure an administrative ID for the active user registry, the validation fails.
5. **Optional:** You can configure and change your External Authorization provider to either WebSphere Authorization, SAF Authorization, or an external JACC provider. For more information, see and "Enabling an external JACC provider" on page 1368. To change the Authorization provider, click **Security > Secure Administration, applications, and infrastructure > External Authorization providers**.
6. Configure the authentication mechanism.

   Configure Lightweight Third-Party Authentication (LTPA), which is the default authentication mechanism, on the Authentication mechanisms and expiration panel. LTPA credentials can be forwarded to other machines. For security reasons, credential expire; however, you can configure the expiration dates on the console. LTPA credentials enable browsers to visit different product servers, which means you do not have to authenticate multiple times. For more information, see "Configuring the Lightweight Third Party Authentication mechanism" on page 1235

   Note: You can configure Simple WebSphere Authentication Mechanism (SWAM) as your authentication mechanism. However, SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. SWAM credentials are not forwardable to other machines and for that reason do not expire. To use SWAM, select the **Use SWAM-no authenticated communication between servers** option.
7. **Optional:** Import and export the LTPA keys for cross-cell single Sign-on (SSO) between cells. For more information, see the following articles:
   - "Exporting Lightweight Third Party Authentication keys" on page 1240.
   - "Importing Lightweight Third Party Authentication keys" on page 1240
8. Configure the authentication protocol for special security requirements from Java clients, if needed.

   You can configure Common Secure Interoperability Version 2 (CSIv2) through links on the Secure administration, applications, and infrastructure panel. The Security Authentication Service (SAS) protocol is provided for backwards compatibility with previous product releases, but is deprecated. Links to the SAS protocol panels display on the Secure administration, applications, and infrastructure panel if your environment contains servers that use previous versions of WebSphere Application Server and support the SAS protocol. For details on configuring CSIv2 or SAS, see the article, "Configuring RMI over IIOP" on page 1302.

   Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

   Attention: V6.0.x IBM no longer ships or supports the Secure Authentication Service (SAS) IIOP security protocol. It is recommended that you use the Common Secure Interoperability version 2 (CSIv2) protocol.
9. Modify or a create a default Secure Sockets Layer (SSL) configuration. This action protects the integrity of the messages sent across the Internet. The product provides a single location where you can specify SSL configurations that the various WebSphere Application Server features that use SSL can utilize, including the LDAP registry, Web container and the authentication protocol (CSIv2 and SAS). For more information, see "Creating a Secure Sockets Layer configuration" on page 1429. After you modify a configuration or create a new configuration, specify it on the **SSL configurations** panel. To get to the SSL configurations panel, complete the following steps:
   a. Click **Security > SSL certificate and key management**.
   b. Under Configuration settings, click **Manage endpoint security configurations >** *configuration_name*.

c. Under Related items, click **SSL configurations**.

You can either edit the DefaultSSLConfig file or create a new SSL configuration with a new alias name. If you create a new alias name for your new keystore and truststore files, change every location that references the DefaultSSLConfig SSL configuration alias. The following list specifies the locations of where the SSL configuration repertoire aliases are used in the WebSphere Application Server configuration.

For any transports that use the new network input/output channel chains, including HTTP and Java Message Service (JMS), you can modify the SSL configuration repertoire aliases in the following locations for each server:

- Click **Server > Application server >** *server_name*. Under Communications, click **Ports**. Locate a transport chain where SSL is enabled and click **View associated transports**. Click *transport_channel_name*. Under Transport Channels, click **SSL Inbound Channel (SSL_2)**.

For the Object Request Broker (ORB) SSL transports, you can modify the SSL configuration repertoire aliases in the following locations. These configurations are for the server-level for WebSphere Application Server and WebSphere Application Server Express and the cell level for WebSphere Application Server Network Deployment.

- Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 inbound transport**.
- Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 outbound transport**.
- V6.0.x Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **SAS inbound transport**
- V6.0.x Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **SAS outbound transport**

For the SOAP Java Management Extensions (JMX) administrative transports, you can modify the SSL configurations repertoire aliases by clicking **Servers > Application servers >** *server_name*. Under Server infrastructure, click **Administration > Administration services**. Under Additional properties, click **JMX connectors > SOAPConnector**. Under Additional properties, click **Custom properties**. If you want to point the sslConfig property to a new alias, click **New** and type sslConfig in the name field, and its value in the Value field.

For the Lightweight Directory Access Protocol (LDAP) SSL transport, you can modify the SSL configuration repertoire aliases by clicking **Security > Secure administration, applications, and infrastructure**. Under User account repository, click the **Available realm definitions** drop-down list, and select **Standalone LDAP registry**.

10. Click **Security > Secure administration, applications, and infrastructure** to configure the rest of the security settings and enable security. For information about these settings, see “Secure administration, applications, and infrastructure settings” on page 1072.
11. Validate the completed security configuration by clicking **OK** or **Apply**. If problems occur, they display at the top of the console page in red type.
12. If there are no validation problems, click **Save** to save the settings to a file that the server uses when it restarts. Saving writes the settings to the configuration repository.

    Important: If you do not click **Apply** or **OK** in the Secure administration, applications, and infrastructure panel before you click **Save**, your changes are not written to the repository. The server must be restarted for any changes to take effect when you start the administrative console.
13. Start the WebSphere Application Server administrative console.

    If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password. This ID is typically the server user ID that is specified when you configured the user registry.

Administrative security

Administrative security determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults. Proper planning is required because incorrectly enabling administrative security can lock you out of the administrative console or cause the server to end abnormally.

Administrative security can be thought of as a ″big switch″ that activates a wide variety of security settings for WebSphere Application Server. Values for these settings can be specified, but they will not take effect until administrative security is activated. The settings include the authentication of users, the use of Secure Sockets Layer (SSL), and the choice of user account repository. In particular, application security, including authentication and role-based authorization, is not enforced unless administrative security is active. Administrative security is enabled by default.

Administrative security represents the security configuration that is effective for the entire security domain. A *security domain* consists of all of the servers that are configured with the same user registry *realm* name. In some cases, the realm can be the machine name of a local operating system registry. In this case, all of the application servers must reside on the same physical machine. In other cases, the realm can be the machine name of a standalone Lightweight Directory Access Protocol (LDAP) registry.

The basic requirement for a security domain is that the access ID that is returned by the registry or repository from one server within the security domain is the same access ID as that returned from the registry or repository on any other server within the same security domain. The *access ID* is the unique identification of a user and is used during authorization to determine if access is permitted to the resource.

The administrative security configuration applies to every server within the security domain.

Why turn on administrative security?

Turning on administrative security activates the settings that protect your server from unauthorized users. Administrative security is enabled by default during the profile creation time. There might be some environments where no security is needed such as a development system. On these systems you can elect to disable administrative security. However, in most environments you should keep unauthorized users from accessing the administrative console and your business applications. Administrative security must be enabled to restrict access.

What does administrative security protect?

The configuration of administrative security for a security domain involves configuring the following technologies:

- Authentication of HTTP clients
- Authentication of IIOP clients
- Administrative console security
- Naming security
- Use of SSL transports
- Role-based authorization checks of servlets, enterprise beans, and mbeans
- Propagation of identities (RunAs)
- The common user registry
- The authentication mechanism
- Other security information that defines the behavior of a security domain includes:
  - The authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security)
  - Other miscellaneous attributes

Fine-grained administrative security:

In releases prior to WebSphere Application Server version 6.1, users granted administrative roles could administer all of the resource instances under the cell. WebSphere Application Server is now more fine-grained, meaning that access can be granted to each user per resource instance.

For example, users can be granted configurator access to a specific instance of a resource only (an application, an application server or a node). Users cannot access any other resources outside of the resources assigned to them. The administrative roles are now per resource instance rather than to the entire cell. However, there is a cell-wide authorization group for backward compatibility. Users assigned to administrative roles in the cell-wide authorization group can still access all of the resources within the cell.

To achieve this instance-based security or fine-grained security, resources that require the same privileges are placed in a group called the *administrative authorization group* or *authorization group*. Users can be granted access to the authorization group by assigning to them the required administrative role.

Fine-grained administrative security can also be used in single-server environments. Various applications in the single server can be grouped and placed in different authorization groups. Therefore, there are different authorization constraints for different applications. Note that the server itself cannot be part of any authorization group.

The AdminSecurityManager role is available for wsadmin users. When using **wsadmin**, users granted this role can map users to administrative roles. Also, when fine grained admin security is used, users granted this role can manage authorization groups. See “Administrative roles and naming service authorization” on page 1334 for detailed explanations of all administrative roles.

There are several administrative security commands that can be used to create authorization groups, map resources to authorization groups, and to assign users to administrative roles within the authorization groups. Following are some examples:

- **Create a new authorization group:**

  ```
  $AdminTask createAuthorizationGroup {-authorizationGroupName authGroup1}
  ```

- **Deleting an authorization group:**

  ```
  $AdminTask deleteAuthorizationGroup {-authorizationGroupName groupName}
  ```

- **Add resources to an authorization group:**

  ```
  $AdminTask addResourceToAuthorizationGroup
  {-authorizationGroupName groupName -resourceName Application=app1}
  ```

- **Remove resources from an authorization group:**

  ```
  $AdminTask removeResourceFromAuthorizationGroup
  {-authorizationGroupName groupName -resourceName Application=app1}
  ```

- **Add user IDs to roles in an authorization group:**

  ```
  $AdminTask mapUsersToAdminRole {-authorizationGroupName groupName
  -roleName administrator -userids user1}
  ```

- **Add group IDs to roles in an authorization group:**

  ```
  $AdminTask mapGroupsToAdminRole {-authorizationGroupName groupName
  -roleName administrator -groupids group1}
  ```

- **Remove user IDs from roles in an authorization group:**

  ```
  AdminTask removeUsersFromAdminRole {-authorizationGroupName
  groupName -roleName administrator -userids user1}
  ```

- **Remove group IDs from roles in an authorization group:**

  ```
  $AdminTask removeGroupsFromAdminRole {-authorizationGroupName
  groupName -roleName administrator -groupids group1}
  ```

Resources that can be added to an authorization group

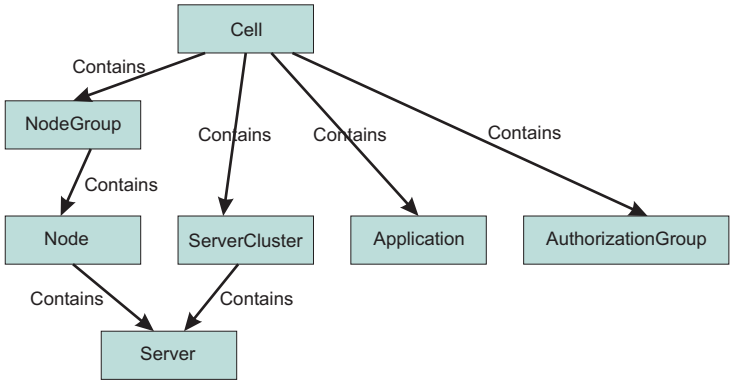
You can add only resource instances of the following types to an authorization group:

- Cell
- Node
- ServerCluster
- Server
- Application
- NodeGroup

If a resource instance is not one of the types listed above, its parent resource will be used.

A resource instance can only belong to one authorization group. However, there is a containment relationship among resource instances. If a parent resource belongs to a different authorization group than that of its child resource instance, the child resource instance implicitly will belong to multiple authorization groups. You cannot add the same resource instance to more than one authorization group.

The following diagram shows the containment relationship among resource instances:



The privileges required for actions on resource instances depend on two factors:

- The authorization group of the administrative resource instance. If a user is granted access to an authorization group, all of the resource instances in that group will be included.
- The containment relationship of the resource instance. If a user is granted access to a parent resource instance, all of the children resource instances will be included.

The privileges required to access various administrative resource instances are shown in the following table:

Resource	Action	Required roles
Server	Start, stop, runtime operations	Server-operator, node-operator, cell-operator
Server	New, delete	Node-configurator, cell-configurator
Server	Edit configuration	Server-configurator, node-configurator, cell-configurator
Server	View configuration, runtime status	Server-monitor, node-monitor, cell-monitor
Node	Restart, stop, sync	Node-operator, Cell-operator
Node	Add, delete	Cell-configurator
Node	Edit configuration	Node-configurator, cell-configurator
Node	View configuration, runtime status	Node-monitor, cell-monitor
Cluster	Start, stop, runtime operations	Cluster-operator, cell-operator

Resource	Action	Required roles
Cluster	New, delete	Cell-configurator
Cluster	Edit configuration	Cluster-configurator, cell-configurator
Cluster	View configuration, runtime status	Cluster-monitor, cell-monitor
Cluster member	Start, stop, runtime operations	Server-operator, cluster-operator, node-operator, cell-operator
Cluster member	New, delete	Node-configurator, cell-configurator
Cluster member	Edit configuration	Server-configurator, cluster-configurator, node-configurator, cell-configurator
Cluster member	View configuration, runtime status	Server-monitor, cluster-monitor, node-monitor, cell-monitor
Application	Start, stop, runtime operations	Application-deployer, cell-operator
Application	Install, uninstall	Cell-configurator application-deployer
Application	Edit configuration	Application-deployer, cell-configurator
Application	View configuration, runtime status	Application-monitor, cell-monitor
Node, cluster	Add, delete	Cell-configurator

The *server-operator* role is the operator role of the authorization group to which the server instance is part of. Similarly, the *node-operator* role is in the operator role of the authorization group to which the node instance is part of.

Fine-grained administrative security is only available for wsadmin users. It is not available for administrative console users. To log in to the administrative console, a user should be granted a monitor role at the cell level at minimum. However, to login using wsadmin, a user should be granted a monitor role for any authorization group.

If you log in to the administrative console as a cell-level administrator, operator, monitor or configurator, you can perform all operations. However, if you want to use additional administrator roles (such as deployer or AdminSecurityManager), or give users access only to specific authorization groups or permissions to non-cell authorizations groups, you must use **wsadmin**.

Fine-grained administrative security in heterogeneous and single-server environments:

Fine-grained administrative security can be used in heterogeneous or single-server environments with some restrictions.

Fine-grained administrative security in a heterogeneous environment

Fine-grained administrative security in a heterogeneous environment has the following restrictions:

- Only nodes that are running WebSphere Application Server Version 6.1 can be part of an administrative authorization group.
- Only servers that are running in a WebSphere Application Server Version 6.1 node can be part of an administrative authorization group.
- Only applications that are targeted on servers running on WebSphere Application Server Version 6.1 can be part of an administrative authorization group.
- If a cluster spans nodes of multiple releases, it cannot be part of an administrative authorization group.
- If a cluster spans nodes of multiple releases, none of its members can be part of an administrative authorization group.
- If an application is targeted on a cluster that spans multiple releases, that application cannot be part of an administrative authorization group.

Fine-grained administrative security in a single-server environment

You can also use fine-grained administrative security in a single-server environment. Various applications in the single server can be grouped and placed in different authorization groups. Therefore, different authorization constraints might exist for different applications.

Life cycle of fine-grained administrative resource

An administrative resource that was once part of an authorization group continues to be part of that authorization group until one of the following events occurs:

- The administrative resource is removed from the authorization group. In this instance, the administrative resource belongs to the cell-level authorization group.
- The administrative resource is removed from the configuration. In this instance, the administrative resource does not exist in the configuration, but still exists in the authorization group. Remove this administrative resource from the authorization group.

After the administrative resource is removed from the authorization group, the administrative authorizer runtime must be notified by using the AuthorizationManager refreshAll MBean method.

The **refreshAll** command must be invoked after AdminConfig.save() and sync nodes. For example:

JACL:

```
// get AuthorizationGroup Mbean
wsadmin> set agBean [$AdminControl queryNames
Type=AuthorizationGroupManager,process=dmgr,*]
```

JYTHON:

```
// get AuthorizationGroup Mbean
wsadmin> set agBean [$AdminControl queryNames
Type=AuthorizationGroupManager,process=dmgr,*]
```

Fine-grained administrative security scenarios:

The following scenarios describe the use of fine-grained administrative security, particularly the new deployment role.

Deployment role scenario 1

In the following scenario, there are four applications configured on server S1, as shown in the following table. Each application must be isolated so that the administrator of one application cannot modify another application. Assume that only user1 can manage application A1, user2 can manage applications A2 and A3, and only user3 can manage application A4.

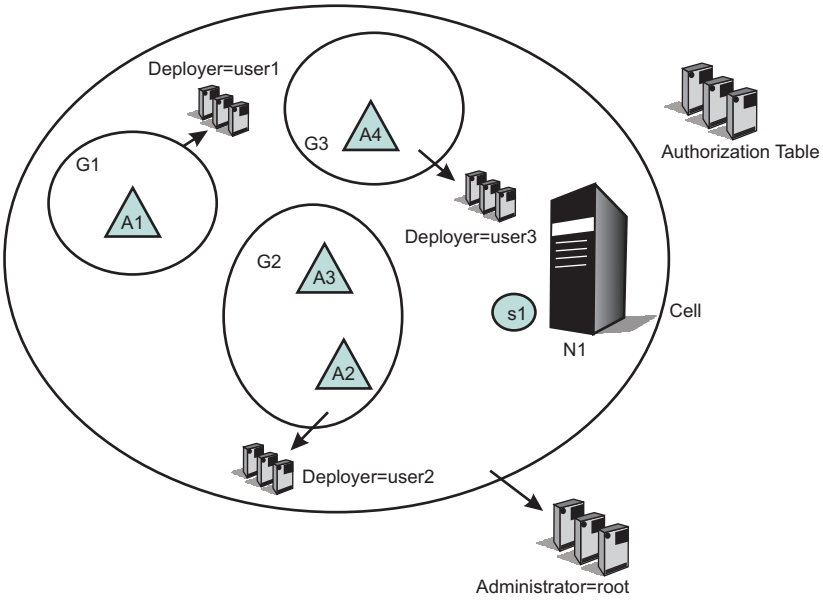
Note: It is not recommended to have an application in one group and its target server in another group. However, that is not always possible. It is common to have many applications on one server. It is still sometimes necessary to isolate the administration of applications running on the same server.

One example is an Application Service Provider (ASP), where a single application server can have multiple vendor applications. In this instance the server administrator is responsible for installing all of the vendor applications. Once applications are installed, each vendor can manage their own application without interfering with other vendor's applications.

Application	Server	Node
A1	S1	N1
A2	S1	N1

Application	Server	Node
A3	S1	N1
A4	S1	N1

We can configure authorization groups as shown in the diagram below:



In the diagram, application A1 is in authorization group G1, applications A2 and A3 are in authorization group G2, and application A4 is in authorization group G3.

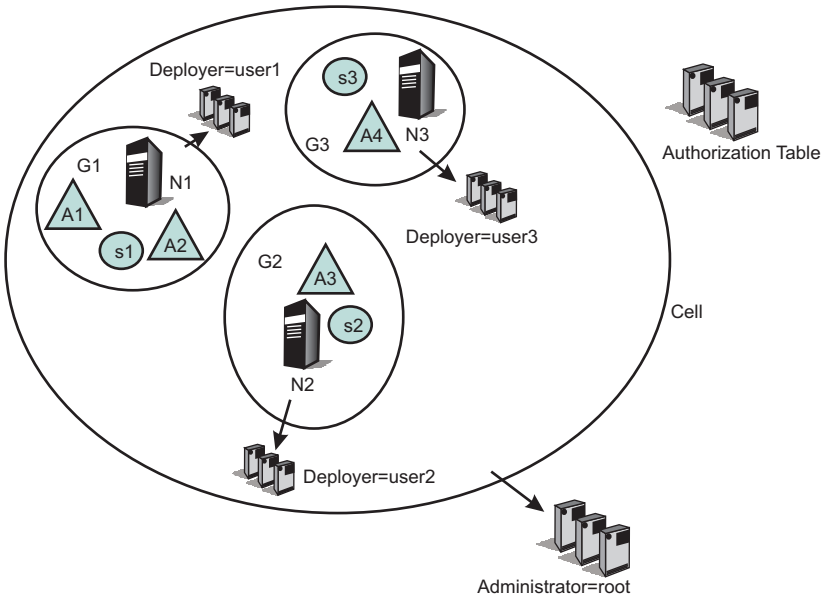
A deployer role is assigned from authorization group G1 to user1, from authorization group G2 to user2, and from authorization group G3 to user3.

Consequently, user1 can perform all of the operations on application A1, user2 on applications A2 and A3, and user3 on application A4. Since all applications share the same server, we cannot put the same server on all authorization groups. Only a cell-level administrator can install an application. After the installation of an application is complete, the deployer of each application can modify their own. To start and stop the server, cell-level administrative authority is required. This type of scenario is useful in an ASP environment.

Deployment role scenario 2

In the following scenario, a group of applications require the same administrative roles to one server. In this example, applications A1 and A2 are related applications, and can be administrated by one set of administrators. They are running on the same server (S1). Applications A3 and A4 require a different set of administrators, and are running on servers S2 and S3 respectively.

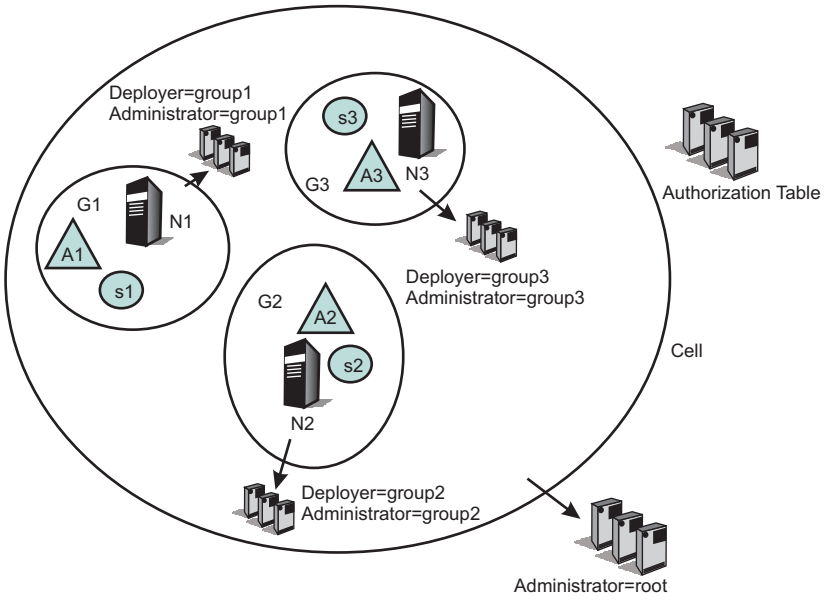
Application	Server	Node
A1	S1	N1
A2	S1	N1
A3	S2	N2
A4	S3	N3



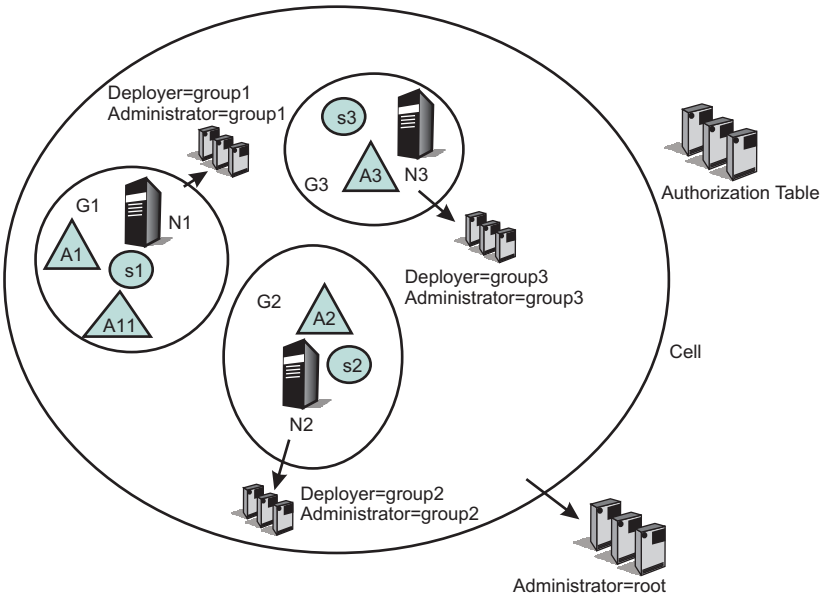
Scenarios that can be applied directly in customer environments

Each developer must be able to modify the configuration for their server, and they must be able to install their application onto that server. They also must be able to start and stop the server as well as the application on the server.

Developers also must be able to configure the server so that they can debug any problems they run into. They must have the ability to update or modify the application being developed. The administrative authorization group for this developer includes at least one server and any applications that the developer installs on that server.



In the following example, developers of authorization group G1 have a new application (A11). They can install and target that new application only on servers within authorization group G1. Also, they can place that new application in their authorization group (G1).



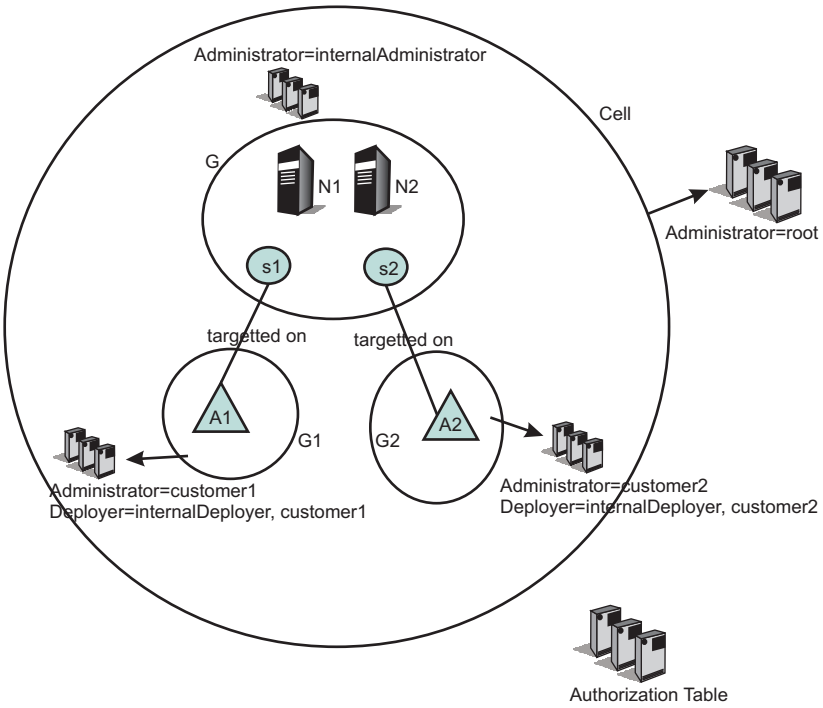
ASP environment scenario

In this scenario, the customer is an ASP. They have their own customers to whom they provide application serving function. They want to enable their customers to administer and monitor their applications, but not to see or administer applications for different customers. In this example, however, the ASP has internal staff administrators whose job it is to maintain the servers.

This internal ASP staff administrator might need to move an application from one server to another to ensure that an application remains available. The internal ASP staff administrator should be able to stop and start the servers and to change their configuration.

In contrast, the ASP customer administrator should not be able to stop or start servers. However, the ASP customer administrator should be able to update their applications running on those servers. The administrative authorization group for the internal ASP administrator can be the whole cell or can include a subset of servers, nodes, clusters and applications. The administrative authorization group for the customer administrator only includes those applications that the customer has paid to have served by this ASP.

The following diagram contains a scenario where two different customers have two different type of applications, and can manage their own applications. However, the servers and nodes on which the applications are running are isolated from their customers. The servers and nodes can only be maintained by the internal administrators. In addition, the customers cannot target their applications on a different server. This can only be performed by the internal administrator or internal deployers.

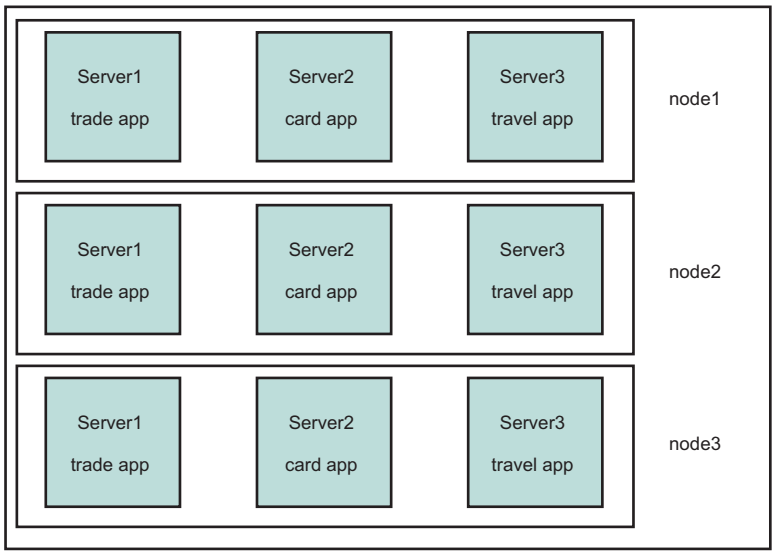


Regional organization scenario

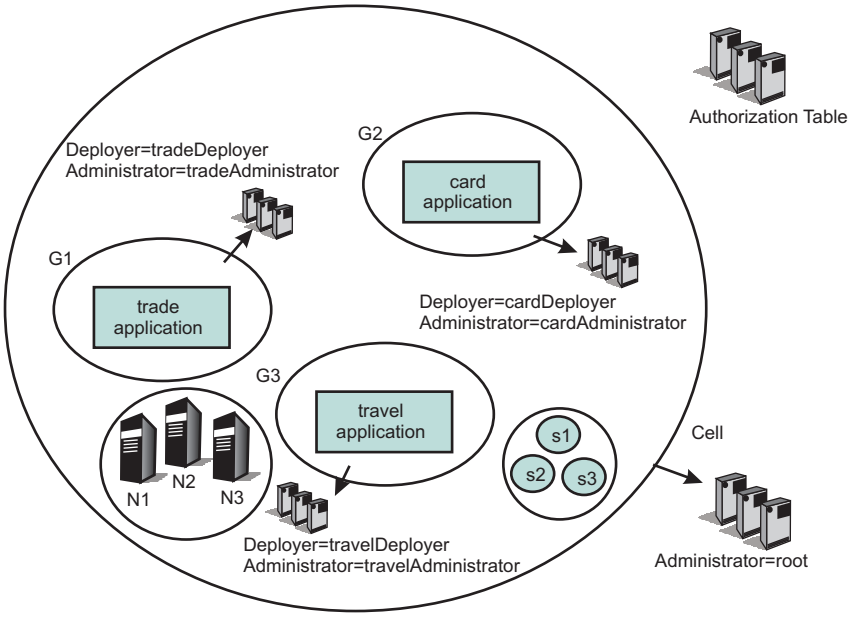
In this scenario, the customer is a large global company. The company's nodes and servers are organized so as to provide application serving for different regions (or alternatively, different lines of business). They want representatives from the different regional areas to be able to monitor and administer the nodes and servers associated with that region. However, they do not want the regional administer to be able to effect any node and server associated with a different region.

The administrative authorization group for each regional representative includes the nodes, servers, clusters and applications associated with that region.

For example, consider a company that provides multiple services, such as a financial institution that provides services like credit card accounts, brokerage accounts, banking accounts, or travel accounts. Each of these services can be separate applications, and the administrator for each of these applications must also be different. The following figure shows one way to configure such a system:

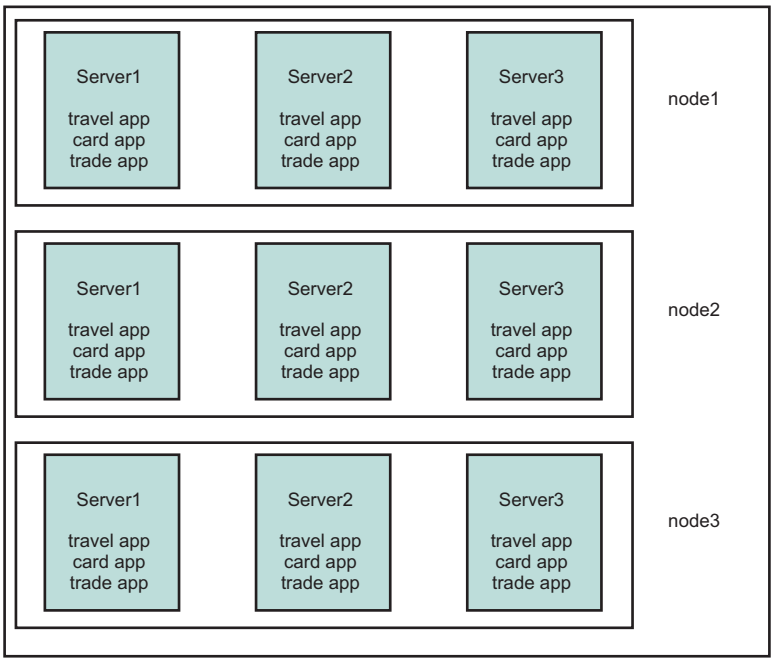


The following figure shows how the resources in such a system can be grouped to isolate administrators from each other:

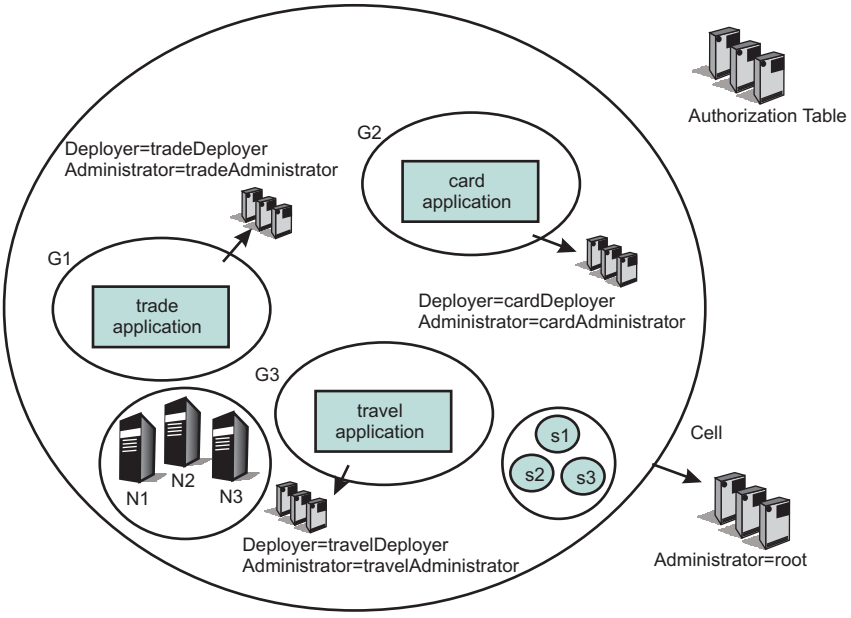


Note that the nodes are not part of any authorization group. Therefore, a trade application administrator cannot stop a server on any of the nodes, and is prevented from stopping a travel application.

The same system can be configured in another way as shown below:



The following figure shows how the resources in such a system can be grouped to isolate administrators from each other:



Application security

Application security enables security for the applications in your environment. This type of security provides application isolation and requirements for authenticating application users

In previous releases of WebSphere Application Server, when a user enabled global security, both administrative and application security were enabled. In WebSphere Application Server Version 6.1, the previous notion of global security is split into administrative security and application security, each of which you can enable separately.

As a result of this split, WebSphere Application Server clients must know whether application security is disabled at the target server. Administrative security is enabled, by default. Application security is disabled, by default. To enable application security, you must enable administrative security. Application security is in effect only when administrative security is enabled.

An Application Server Enablement Tag, which is specific to WebSphere Application Server, is imported into the Interoperable Object Reference (IOR) to indicate if application security is disabled for the server where the object lives. This tag is server-specific and enables clients to know when application security is disabled at the target server of its request.

For Web resources, when application security is enabled, authentication is prompted for lookups on the application server.

For enterprise bean resources, when application security is disabled, the client Common Secure Interoperability version 2 (CSIv2) code ignores the CSIv2 security tags for objects that are unknown system objects. When pure clients see that application security is disabled, these clients prompt for naming lookups, but do not prompt for enterprise bean operations.

Java 2 security

Java 2 security provides a policy-based, fine-grain access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 security guards access to system resources such as file I/O, sockets, and properties. Java 2 Platform, Enterprise Edition (J2EE) security guards access to Web resources such as servlets, JavaServer Pages (JSP) files and Enterprise JavaBeans (EJB) methods.

WebSphere Application Server security includes the following technologies:

- Java 2 Security Manager
- Java Authentication and Authorization Service (JAAS)
- Java 2 Connector authentication data entries
- V6.0.x Common Secure Interoperability Version 2 (CSIv2) or Security Authentication Service (SAS)

  Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.
- J2EE role-based authorization
- Secure Sockets Layer (SSL) configuration

Because Java 2 security is relatively new, many existing or even new applications might not be prepared for the very fine-grain access control programming model that Java 2 security is capable of enforcing. Administrators need to understand the possible consequences of enabling Java 2 security if applications are not prepared for Java 2 security. Java 2 security places some new requirements on application developers and administrators.

Java 2 security for deployers and administrators

Although Java 2 security is supported, it is disabled by default. You can configure Java 2 security and administrative security independently of one another. Disabling administrative security does not disable Java 2 security automatically. You need to explicitly disable it.

If your applications, or third-party libraries are not ready, having Java 2 security enabled causes problems. You can identify these problems as Java 2 security AccessControlExceptions in the system log or trace files. If you are unsure about the Java 2 security readiness of your applications, disable Java 2 security initially to get your application installed and verify that it is working properly.

Implications exist if Java 2 security is enabled; deployers or administrators are required to make sure that all the applications are granted the required permissions; otherwise, applications might fail to run. By default, applications are granted the permissions that are recommended in the J2EE 1.4 Specification. For the details of default permissions granted to applications in the product, refer to the following policy files:

- *app_server_root*/java/jre/lib/security/java.policy
- *app_server_root*/properties/server.policy
- *profile_root*/config/cells/*cell_name*/nodes/*node_name*/app.policy

The policy embodied by these policy files cannot be made more restrictive because the product might not have the necessary Java 2 security doPrivileged APIs in place. The restrictive policy is the default policy. You can grant additional permissions, but you cannot make the default more restrictive because AccessControlExceptions exceptions are generated from within WebSphere Application Server. The product does not support a more restrictive policy than the default that is defined in the policy files previously mentioned.

Several policy files are used to define the security policy for the Java process. These policy files are static (code base is defined in the policy file) and in the default policy format provided by the IBM Developer Kit, Java Technology Edition. For enterprise application resources and utility libraries, WebSphere Application Server provides dynamic policy support. The code base is dynamically calculated based on deployment information and permissions are granted based on template policy files during runtime. Refer to the “Java 2 security policy files” on page 1065 for more information.

Syntax errors in the policy files cause the application server process to fail, so edit these policy files carefully.

If an application is not prepared for Java 2 security, if the application provider does not provide a `was.policy` file as part of the application, or if the application provider does not communicate the expected permissions the application is likely to cause Java 2 security access control exceptions at runtime. It might not be obvious that an application is not prepared for Java 2 security. Several run-time debugging aids help troubleshoot applications that might have access control exceptions. See the Java 2 security debugging aids for more details. See “Handling applications that are not Java 2 security ready” on page 1064 for information and strategies for dealing with such applications.

It is important to note when Java Security is enabled in the administrative security settings, the installed security manager does not currently check modifyThread and modifyThreadGroup permissions for non-system threads. Allowing Web and Enterprise JavaBeans (EJB) application code to create or modify a thread can have a negative impact on other components of the container and can affect the capability of the container to manage enterprise bean life cycles and transactions.

Java 2 security for application developers

Application developers must understand the permissions that are granted in the default WebSphere policy and the permission requirements of the SDK APIs that their application calls to know whether additional permissions are required. The Permissions in the Java 2 SDK reference in the resources section describes which APIs require which permission.

Application providers can assume that applications have the permissions granted in the default policy previously mentioned. Applications that access resources not covered by the default WebSphere policy are required to grant the additional Java 2 security permissions to the application.

While it is possible to grant the application additional permissions in one of the other dynamic WebSphere policy files or in one of the more traditional `java.policy` static policy files, the `was.policy` file, which is embedded in the EAR file ensures the additional permissions are scoped to the exact application that requires them. Scoping the permission beyond the application code that requires it can permit code that normally does not have permission to access particular resources.

If an application component is being developed, like a library that might actually be included in more than one `.ear` file, then the library developer needs to document the required Java 2 permissions that are required by the application assembler. There is no `was.policy` file for library-type components. The developer must communicate the required permissions through application programming interface (API) documentation or some other external documentation.

If the component library is shared by multiple enterprise applications, the permissions can be granted to all enterprise applications on the node in the `app.policy` file.

If the permission is only used internally by the component library and the application is never granted access to resources that are protected by the permission, it might be necessary to mark the code as **privileged**. Refer to the, AccessControlException, topic for more details. However, improperly inserting a doPrivileged call might open up security holes. Understand the implication of doPrivileged call to make a correct judgement.

The section on Dynamic policy files in “Java 2 security policy files” on page 1065 describes how the permissions in the `was.policy` files are granted at runtime.

Developing an application to use with Java 2 security might be a new skill and impose a security awareness not previously required of application developers. Describing the Java 2 security model and the implications on application development is beyond the scope of this section. The following URL can help you get started: http://java.sun.com/j2se/1.5.0/docs/guide/security/index.html.

Debugging Aids

The WebSphere Application Server SYSOUT file and the com.ibm.websphere.java2secman.norethrow property are the two primary aids for debugging.

The WebSphere System Log or Trace Files

The AccessControl exception that is logged in the system log or trace files contains the permission violation that causes the exception, the exception call stack, and the permissions granted to each stack frame. This information is usually enough to determine the missing permission and the code requiring the permission.

The com.ibm.websphere.java2secman.norethrow property

When Java 2 security is enabled in WebSphere Application Server, the security manager component creates a java.security.AccessControl exception when a permission violation occurs. This exception, if not handled, often causes a run-time failure. This exception is also logged in the SYSOUT file.

However, when the Java virtual machine com.ibm.websphere.java2secman.norethrow property is set and has a value of `true`, the security manager does not create the AccessControl exception. This information is logged.

To set the com.ibm.websphere.java2secman.norethrow property for the server, go to the WebSphere Application Server administrative console and click **Servers > Application Servers >** *server_name*. Under Server infrastructure, click **Java and Process Management > Process definition**. Under Additional properties, click **Java Virtual Machine > Custom Properties > New**. In the **Name** field, type `com.ibm.websphere.java2secman.norethrow`. In the **Value** field, type `true`.

This property is intended for a sandbox or debug environment because it instructs the security manager not to create the AccessControl exception. Java 2 security is not enforced. Do not use this property in a production environment where a relaxed Java 2 security environment weakens the integrity that Java 2 security is intended to produce.

This property is valuable in a sandbox or test environment where the application can be thoroughly tested and where the system log or trace files can be inspected for AccessControl exceptions. Because this property does not create the AccessControl exception, it does not propagate the call stack and does not cause a failure. Without this property, you have to find and fix AccessControl exceptions one at a time.

Handling applications that are not Java 2 security ready

If the increased system integrity that Java 2 security provides is important, then contact the application provider to have the application support Java 2 security or at least communicate the required additional permissions beyond the default WebSphere Application Server policy that must be granted.

The easiest way to deal with such applications is to disable Java 2 security in WebSphere Application Server. The downside is that this solution applies to the entire system and the integrity of the system is not as strong as it might be. Disabling Java 2 security might not be acceptable depending on the organization security policies or risk tolerances.

Another approach is to leave Java 2 security enabled, but to grant either just enough additional permissions or grant all permissions to just the problematic application. Granting permissions however, might not be a trivial thing to do. If the application provider has not communicated the required permissions in some way, no easy way exists to determine what the required permissions are and granting all permissions might be the only choice. You minimize this risk by locating this application on a different node, which might help isolate it from certain resources. Grant the java.security.AllPermission permission in the `was.policy` file that is embedded in the application `.ear` file, for example:

```
grant codeBase "file:${application}" {
            permission java.security.AllPermission;
        };
```

The server.policy file

The `server.policy` file is located in the *app_server_root*`/properties/` directory.

This policy defines the policy for the WebSphere Application Server classes. At present, all the server processes on the same installation share the same `server.policy` file. However, you can configure this file so that each server process can have a separate `server.policy` file. Define the policy file as the value of the java.security.policy Java system properties . For details of how to define Java system properties, refer to the Process definition section of the Manage application servers file.

The `server.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to other machines. Use the `server.policy` file to define Java 2 security policy for server resources. Use the `app.policy` file (per node) or the `was.policy` file (per enterprise application) to define Java 2 security policy for enterprise application resources.

The java.policy file

The file represents the default permissions that are granted to all classes. The policy of this file applies to all the processes launched by the Java Virtual Machine in the WebSphere Application Server.

The `java.policy` file is located in the *app_server_root*`/java/jre/lib/security` directory.

Troubleshooting

Symptom:

`Error message CWSCJ0314E`: Current Java 2 security policy reported a potential violation of Java 2 security permission. Refer to Problem Determination Guide for further information.{0}Permission\:{1}Code\ :{2}{3}Stack Trace\:{4}Code Base Location\:{5} Current Java 2 security policy reported a potential violation

of Java 2 Security Permission. Refer to Problem Determination Guide for further information.{0}Permission\
:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}

Problem:

The Java security manager checkPermission method reported a security exception on the subject permission with debugging information. The reported information can be different with respect to the system configuration. This report is enabled by either configuring a Reliability Availability Service Ability (RAS) trace into debug mode or specifying a Java property.

See Enabling trace for information on how to configure RAS trace in debug mode.

Specify the following property in the JVM Settings panel from the administrative console:
java.security.debug. Valid values include:
access
Print all debug information including: required permission, code, stack, and code base location.
stack Print debug information including: required permission, code, and stack.
failure Print debug information including: required permission and code.

Recommended response:

The reported exception might be critical to the secure system. Turn on security trace to determine the potential code that might have violated the security policy. After the violating code is determined, verify if the attempted operation is permitted with respect to Java 2 security, by examining all applicable Java 2 security policy files and the application code.

If the application is running with Java Mail, this message might be benign. You can update the `was.policy` file to grant the following permissions to the application:

```
permission  java.io.FilePermission  ″${user.home}${/}.mailcap″,  ″read″;
permission  java.io.FilePermission  ″${user.home}${/}.mime.types″,  ″read″;
permission  java.io.FilePermission  ″${java.home}${/}lib${/}mailcap″,  ″read″;
permission  java.io.FilePermission  ″${java.home}${/}lib${/}mime.types″,  ″read″;
```

Messages

Message:	`CWSCJ0313E`: Java 2 security manager debug message flags are initialized\: TrDebug: {0}, Access: {1}, Stack: {2}, Failure: {3}
Problem:	Configured values of the valid debug message flags for security manager.

Message:	`CWSCJ0307E`: Unexpected exception is caught when trying to determine the code base location. Exception: {0}
Problem:	An unexpected exception is caught when the code base location is determined.

Java 2 security policy files:

The Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 and later specifications have a well-defined programming model of responsibilities between the container providers and the application code. Using Java 2 security manager to help enforce this programming model is recommended. Certain operations are

not supported in the application code because such operations interfere with the behavior and operation of the containers. The Java 2 security manager is used in the product to enforce responsibilities of the container and the application code.

This product provides support for policy file management. A number of policy files in the product are either static or dynamic. *Dynamic policy* is a template of permissions for a particular type of resource. No relative code base is defined in the dynamic policy template. The code base is dynamically calculated from the deployment and run-time data.

Static policy files

Policy file	Location
java.policy	*app_server_root*/java/jre/lib/security/java.policy. Default permissions are granted to all classes. The policy of this file applies to all the processes launched by WebSphere Application Server.
server.policy	*profile_root*/properties/server.policy. Default permissions are granted to all the product servers.
client.policy	*profile_root*/properties/client.policy. Default permissions are granted for all of the product client containers and applets on a node.

The static policy files are not managed by configuration and file replication services. Changes made in these files are local and are not replicated to other nodes in the Network Deployment cell.

Dynamic policy files

Policy file	Location
spi.policy	*profile_root*/config/cells/*cell_name* /nodes/*node_name*/spi.policy This template is for the Service Provider Interface (SPI) or the third-party resources that are embedded in the product. Examples of SPI are the Java Message Service (JMS) in MQ Series and Java database connectivity (JDBC) drivers. The code base for the embedded resources are dynamically determined from the configuration (resources.xml file) and run-time data, and permissions that are defined in the spi.policy files are automatically applied to these resources and JAR files that are specified in the class path of a resource adapter. The default permission of the spi.policy file is java.security.AllPermissions.
library.policy	*profile_root*/config/cells/*cell_name*/nodes /*node_name*/library.policy This template is for the library (Java library classes). You can define a shared library to use in multiple product applications. The default permission of the library.policy file is empty.
app.policy	*profile_root*/config/cells/*cell_name* /nodes/*node_name*/app.policy The app.policy file defines the default permissions that are granted to all of the enterprise applications running on *node_name* in *cell_name*.
was.policy	*profile_root*/config/cells/*cell_name* /applications/*ear_file_name*/deployments/ *application_name*/META-INF/was.policy This template is for application-specific permissions. The was.policy file is embedded in the enterprise archive (EAR) file.
ra.xml	*rar_file_name*/META-INF/was.policy.RAR. This file can have a permission specification that is defined in the ra.xml file. The ra.xml file is embedded in the RAR file.

Grant entries that are specified in the `app.policy` and `was.policy` files must have a code base defined. If grant entries are specified without a code base, the policy files are not loaded properly and the application can fail. If the intent is to grant the permissions to all applications, use *file:${application}* as a code base in the grant entry.

Syntax of the policy file

A policy file contains several policy entries. The following example depicts each policy entry format:

```
grant [codebase <Codebase>] {
permission <Permission>;
 permission <Permission>;
permission <Permission>;
};

<CodeBase>:  A URL.
   For example,  "file:${java.home}/lib/tools.jar"
      When [codebase <Codebase>] is not specified, listed
               permissions are applied to everything.
      If URL ends with a JAR file name,  only the classes in the
               JAR file belong to the codebase.
               If URL ends with "/", only the class files in the specified
               directory belong to the codebase.
               If URL ends with "*", all JAR and class files in the specified
               directory belong to the codebase.
               If URL ends with "-", all JAR and class files in the specified
               directory and its subdirectories belong to the codebase.
<Permissions>: Consists from
       Permission Type   : class name of the permission
          Target Name       : name specifying the target
          Actions           : actions allowed on target
   For example,
         java.io.FilePermission "/tmp/xxx", "read,write"
```

Refer to developer kit specifications for the details of each permission.

Syntax of dynamic policy

You can define permissions for specific types of resources in dynamic policy files for an enterprise application. This action is achieved by using *product-reserved symbols*. The reserved symbol scope depends on where it is defined. If you define the permissions in the `app.policy` file, the symbol applies to all the resources on all of the enterprise applications that run on *node_name*. If you define the permissions in the `META-INF/was.policy` file, the symbol applies only to the specific enterprise application. Valid symbols for the code base are listed in the following table:

Symbol	Meaning
file:${application}	Permissions apply to all the resources within the application
file:${jars}	Permissions apply to all the utility Java archive (JAR) files within the application
file:${ejbComponent}	Permissions apply to the Enterprise JavaBeans (EJB) resources within the application
file:${webComponent}	Permissions apply to the Web resources within the application
file:${connectorComponent}	Permissions apply to the connector resources within the application

You can specify the module name for a granular setting, except for these entries that are specified by the code base symbols. For example:

```
grant codeBase "file:DefaultWebApplication.war" {
   permission java.security.SecurityPermission "printIdentity";
 };

grant codeBase "file:IncCMP11.jar" {
permission java.io.FilePermission
"${user.install.root}${/}bin${/}DefaultDB${/}-",
"read,write,delete";
};
```

The sixth and seventh lines in the previous code sample are one continuous line. You can use a relative code base only in the `META-INF/was.policy` file. Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

Symbol	Meaning
file:${application}	Permissions apply to all the resources within the application
file:${jars}	Permissions apply to all the utility JAR files within the application
file:${ejbComponent}	Permissions apply to the enterprise beans resources within the application
file:${webComponent}	Permissions apply to the Web resources within the application
file:${connectorComponent}	Permissions apply to the connector resources both within the application and in the standalone connector resources.

Five embedded symbols are provided to specify the path and the name for the java.io.FilePermission permission. These symbols enable flexible permission specification. The absolute file path is fixed after the installation of the application.

Symbol	Meaning
${app.installed.path}	Path where the application is installed
${was.module.path}	Path where the module is installed
${current.cell.name}	Current cell name
${current.node.name}	Current node name
${current.server.name}	Current server name

Attention: Do not use the `${was.module.path}` in the `${application}` entry.

Carefully determine where to add a new permission. An incorrectly specified permission causes an AccessControlException exception. Because dynamic policy resolves the code base at runtime, determining which policy file has a problem is difficult. Add a permission only to the necessary resources. For example, `use ${ejbcomponent}`, and `etc` instead of `${application}`, and update the `was.policy` file instead of the `app.policy` file, if possible.

Static policy filtering

Limited static policy filtering support exists. If the `app.policy` file and the `was.policy` file have permissions that are defined in the `filter.policy` file with thefilterMask keyword, the runtime removes the permissions from the applications and an audit message is logged. However, if the permissions that are defined in the `app.policy` and the `was.policy` files are compound permissions, for example, java.security.AllPermission,

the permission is not removed, but a warning message is written to the log file. The policy filtering only supports Developer Kit permissions; the permissions package name begins with `java` or `javax`.

Run-time policy filtering support is provided to force stricter filtering. If the `app.policy` file and the `was.policy` file have permissions that are defined in the `filter.policy` file with the runtimeFilterMask keyword, the runtime removes the permissions from the applications no matter what permissions are granted to the application. For example, even if a `was.policy` file has the java.security.AllPermission permission granted to one of its modules, specified permissions such as the runtimeFilterMask permission are removed from the granted permission during runtime.

If the **Warn if applications are granted custom permissions** option on the Secure administration, applications, and infrastructure panel is enabled and if the `app.policy` file and the `was.policy` file contain custom permissions (non-Developer Kit permissions, where the permissions package name begins with `java` or `javax`), a warning message logs. The permission is not removed. If the AllPermission permission is listed in the `app.policy` file and the `was.policy` file, a warning message logs.

Policy file editing

Using the policy tool that is provided by the Developer Kit (*app_server_root*`/java/jre/bin/policytool`), to edit the previous policy files is recommended. For Network Deployment, extract the policy files from the repository before editing. After the policy file is extracted, use the policy tool to edit the file. Check the modified policy files into the repository and synchronize them with other nodes.

If syntax errors exist in the policy files, the enterprise application or the server process might fail to start. Be cautious when editing these policy files. For example, if a policy has a trailing space in the policy permission target name, the policy fails to parse the permission properly in the IBM Developer Kit, Java Technology Edition Version 5. In the following example, note the space before the last quote: `* \"*\" "`

```
grant {
    permission javax.security.auth.PrivateCredentialPermission
         "javax.resource.spi.security.PasswordCredential * \"*\" ","read";
};
```

If the permission is in a policy file that is loaded by the IBM Developer Kit, Java Technology Edition policy tool Version 5, the following message might display:

```
Errors have occurred while opening the policy configuration.
View the warning log for more information.
```

or the following message might display in a warning log:

```
Warning: Invalid argument(s) for constructor:
javax.security.auth.PrivateCredentialPermission.
```

To fix this problem, edit the permission and remove the trailing space. When the trailing space is removed, the permission loads properly. The following code sample shows the corrected permission:

```
grant {
    permission javax.security.auth.PrivateCredentialPermission
          "javax.resource.spi.security.PasswordCredential * \"*\",read";
}
```

Troubleshooting

To debug the dynamic policy, choose one of three ways to generate the detail report of the AccessControlException exception.

- **Trace** (Configured by RAS trace). Enables traces with the trace specification:

  Attention: The following command is one continuous line

  ```
  com.ibm.ws.security.policy.*=all=enabled:
  com.ibm.ws.security.core.SecurityManager=all=enabled
  ```

- **Trace** (Configured by property). Specifies a Java java.security.debug property. Valid values for the java.security.debug property are as follows:
  - Access. Print all debug information including required permission, code, stack, and code base location.
  - Stack. Print debug information including, required permission, code, and stack.
  - Failure. Print debug information including required permission and code.
- **ffdc**. Enable `ffdc`, modify the `ffdcRun.properties` file by changing `Level=4` and `LAE=true`. Look for an Access Violation keyword in the log file.

Access control exception:

The Java 2 security behavior is specified by its *security policy*. The security policy is an access-control matrix that specifies which system resources certain code bases can access and who must sign them. The Java 2 security policy is declarative and it is enforced by the java.security.AccessController.checkPermission method.

The following example depicts the algorithm for the java.security.AccessController.checkPermission method. For the complete algorithm, refer to the Java 2 security check permission algorithm in Resources for learning.

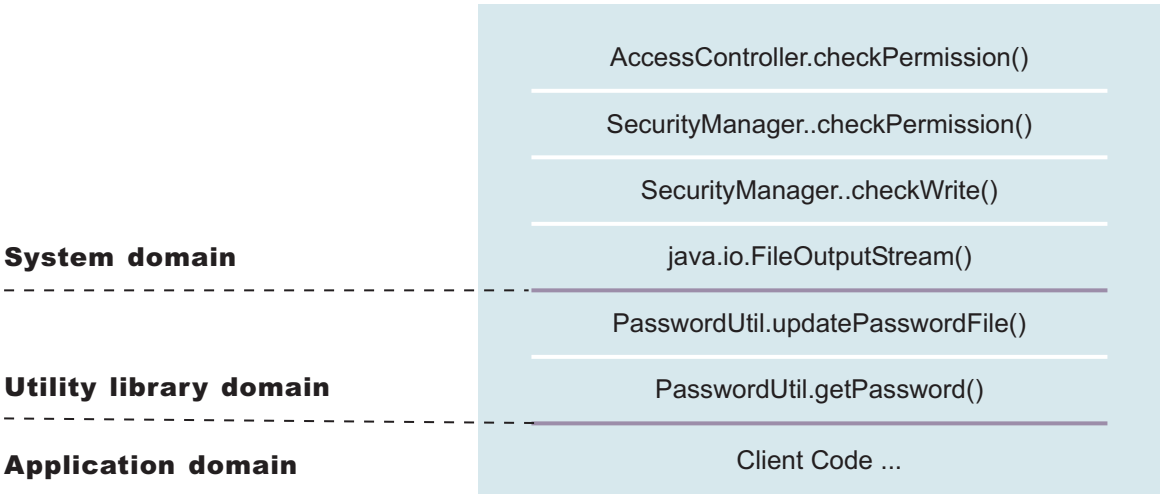
```
i = m;
while (i > 0) {
 if (caller i's domain does not have the permission)
  throw AccessControlException;
 else if (caller i is marked as privileged)
  return;
 i = i - 1;
};
```

The algorithm requires that all the classes or callers on the call stack have the permissions when a java.security.AccessController.checkPermission method is performed or the request is denied and a java.security.AccessControlException exception is created. However, if the caller is marked as *privileged* and the class (caller) is granted these permissions, the algorithm returns and does not traverse the entire call stack. Subsequent classes (callers) do not need the required permission granted.

A java.security.AccessControlException exception is created when certain classes on the call stack are missing the required permissions during a java.security.AccessController.checkPermission method. Two possible resolutions to the java.security.AccessControlException exception are as follows:
- If the application is calling a Java 2 security-protected application programming interface (API), grant the required permission to the application Java 2 security policy. If the application is not calling a Java 2 security-protected API directly, the required permission results from the side-effect of the third-party APIs accessing Java 2 security-protected resources.
- If the application is granted the required permission, it gains more access than it needs. In this case, it is likely that the third party code that accesses the Java 2 security-protected resource is not properly marked as privileged.

Example call stack

This example of a call stack indicates where application code is using a third-party API utility library to update the password. The following example is presented to illustrate the point. The decision of where to mark the code as privileged is application-specific and is unique in every situation. This decision requires great depth of domain knowledge and security expertise to make the correct judgement. A number of well written publications and books are available on this topic. Referencing these materials for more detailed information is recommended.



You can use the **PasswordUtil** utility to change the password of a user. The utility types in the old password and the new password twice to ensure that the correct password is entered. If the old password matches the one stored in the password file, the new password is stored and the password file updates. Assume that none of the stack frame is marked as privileged. According to the java.security.AccessController.checkPermission algorithm, the application fails unless all the classes on the call stack are granted write permission to the password file. The client application does not have permission to write to the password file directly and to update the password file at will.

However, if the PasswordUtil.updatePasswordFile method marks the code that accesses the password file as privileged, then the check permission algorithm does not check for the required permission from classes that call thePasswordUtil.updatePasswordFile method for the required permission as long as the **PasswordUtil** class is granted the permission. The client application can successfully update a password without granting the permission to write to the password file.

The ability to mark code privileged is very flexible and powerful. If this ability is used incorrectly, the overall security of the system can be compromised and security holes can be exposed. Use the ability to mark code privileged carefully.

Resolution to the java.security.AccessControlException exception

As described previously, you have two approaches to resolve a java.security.AccessControlException exception. Judge these exceptions individually to decide which of the following resolutions is best:
1. Grant the missing permission to the application.
2. Mark some code as privileged, after considering the issues and risks.

Enabling security for the realm

Use this topic to enable IBM WebSphere Application Server security. You must enable administrative security for all other security settings to function.

WebSphere Application Server uses cryptography to protect sensitive data and to ensure confidentiality and integrity of communications between WebSphere Application Server and other components in the network. Cryptography is also used by Web services security when certain security constraints are configured for the Web Services application.

WebSphere Application Server uses Java Secure Sockets Extension (JSSE) and Java Cryptography Extension (JCE) libraries in the Software Development Kit (SDK) to perform this cryptography. The SDK provides strong but limited jurisdiction policy files. Unrestricted policy files provide the ability to perform full strength cryptography and to improve performance.

WebSphere Application Server provides a SDK 5 that contains strong, but limited jurisdiction policy files. You can download the unrestricted policy files from the following Web site: IBM developer kit: Security information. Complete the following steps to download and install the new policy files:

1. Click **J2SE 5.0**
2. Scroll down the page then click **IBM SDK Policy files**.

   The Unrestricted JCE Policy files for SDK 5 Web site displays.
3. Click **Sign in** and provide your IBM.com ID and password.
4. Select **Unrestricted JCE Policy files for SDK 5** and click **Continue**.
5. View the license and click **I Agree** to continue.
6. Click **Download Now**.
7. Extract the unlimited jurisdiction policy files that are packaged in the ZIP file. The ZIP file contains a `US_export_policy.jar` file and a `local_policy.jar` file.
8. In your WebSphere Application Server installation, go to the `$JAVA_HOME/jre/lib/security` directory and back up your `US_export_policy.jar` and `local_policy.jar` files.
9. Replace your `US_export_policy.jar` and `local_policy.jar` files with the two files that you downloaded from the IBM.com Web site.
1. Enable security in the WebSphere Application Server. Make sure that all node agents within the cell are active beforehand.

   For more information, see “Enabling security” on page 1047. It is important to click **Security > Secure administration, applications, and infrastructure**. Select an available realm definition from the list, and then click **Set as current** so that security is enabled upon a server restart.

   Note: In previous releases of WebSphere Application Server, the **Set as current** option is known as the **Enable global security** option.
2. Before restarting the server, log off the administrative console. You can log off by clicking **Logout** at the top menu bar.
3. Stop the server by going to the command line in the WebSphere Application Server `/bin` directory and issue a `stopServer` *`server_name`* command.
4. Restart the server in secure mode by issuing the command `startServer` *`server_name`*. Once the server is secure, you cannot stop the server again without specifying an administrative user name and password. To stop the server once security is enabled, issue the command, `stopServer` *`server_name`* `-username` *`user_id`* `-password` *`password`*. Alternatively, you can edit the `soap.client.props` file in the *`profile_root`*`/properties` directory, and edit the `com.ibm.SOAP.loginUserid` or `com.ibm.SOAP.loginPassword` properties to contain these administrative IDs.

   If you have any problems restarting the server, review the output logs in the *`profile_root`*`/logs/server_name` directory. Check the Troubleshooting security configurations article for any common problems.

Secure administration, applications, and infrastructure settings:

Use this page to configure administrative, application, and infrastructure security on a global level.

To view this administrative console page, click **Security > Secure administration, applications, and infrastructure**.

Security has some performance impacts on your applications. The performance impacts can vary depending upon the application workload characteristics. You must first determine that the needed level of security is enabled for your applications, and then measure the impact of security on the performance of your applications.

When security is configured, validate any changes to the user registry or authentication mechanism panels. Click **Apply** to validate the user registry settings. An attempt is made to authenticate the server ID

or to validate the admin ID (if internalServerID is used) to the configured user registry. Validating the user registry settings after enabling administrative security can avoid problems when you restart the server for the first time.

Security configuration wizard:

Launches a wizard that enables you to configure the basic administrative and application security settings. This process restricts administrative tasks and applications to authorized users.

Using this wizard, you can configure application security, resource or Java 2 Connector (J2C) security, and a user registry. You can configure an existing registry and enable administrative, application, and resource security.

When you apply changes made by using the security configuration wizard, administrative security is turned on by default.

Security configuration report:

Launches a security configuration report that displays the core security settings of the application server. The report also displays the administrative users and groups and the CORBA naming roles.

A current limitation to the report is that it does not display application level security information. The report also does not display information on Java Message Service (JMS) security, bus security, or Web Services security.

Enable administrative security:

Specifies whether to enable administrative security for this application server domain. Administrative security requires users to authenticate before obtaining administrative control of the application server.

For more information, see “Administrative roles” on page 1340.

When enabling security, set the authentication mechanism configuration and specify a valid user ID and password (or a valid admin ID when internalServerID feature is used) in the selected registry configuration.

Note: There is a difference between the user ID (which is normally called the admin ID), which identifies administrators who manage the environment, and a server ID, which is used for server-to-server communication. You do not need to enter a server ID and password when you are using the internal server ID feature. However, optionally, you can specify a server ID and password. To specify the server ID and password, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User accounts repository, select the repository and click **Configure**.
3. Specify the server ID and password in the Server user identity section.

Default: Enabled

Enable application security:

Enables application-level security unless the option is overwritten at the server level.

Default: Disabled

Use Java 2 security to restrict application access to local resources:

Specifies whether to enable or disable Java 2 security permission checking. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

When the **Use Java 2 security to restrict application access to local resources** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, the application might fail to run properly until the required permissions are granted in either the app.policy file or the was.policy file of the application. AccessControl exceptions are generated by applications that do not have all the required permissions. See “Java 2 security” on page 1061 for more information about Java 2 security.

Default: Disabled

Disabling security if you have server startup problems:

If your server does not restart after you enable administrative security, you can disable security. Go to your *app_server_root*/bin directory and run the wsadmin -conntype NONE command. At the wsadmin> prompt, enter securityoff and then type exit to return to a command prompt. Restart the server with administrative security disabled to check any incorrect settings through the administrative console.

Warn if applications are granted custom permissions:

Specifies that during application deployment and application start, the security runtime issues a warning if applications are granted any custom permissions. Custom permissions are permissions that are defined by the user applications, not Java API permissions. Java API permissions are permissions in the java.* and javax.* packages.

The application server provides support for policy file management. A number of policy files are available in this product, some of them are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. No code base is defined and no relative code base is used in the dynamic policy template. The real code base is dynamically created from the configuration and run-time data. The filter.policy file contains a list of permissions that you do not want an application to have according to the J2EE 1.4 specification. For more information on permissions, see “Java 2 security policy files” on page 1065.

Important: You cannot enable this option without enabling the **Use Java 2 security to restrict application access to local resources** option.

Default: Disabled

Restrict access to resource authentication data:

Enable this option to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data.

Consider enabling this option when both of the following conditions are true:

- Java 2 security is enforced.
- The application code is granted the accessRuntimeClasses WebSphereRuntimePermission permission in the was.policy file found within the application enterprise archive (EAR) file. For example, the application code is granted the permission when the following line is found in your was.policy file:

```
permission com.ibm.websphere.security.WebSphereRuntimePermission "accessRuntimeClasses";
```

The **Restrict access to resource authentication data** option adds fine-grained Java 2 security permission checking to the default principal mapping of the WSPrincipalMappingLoginModule

implementation. You must grant explicit permission to Java 2 Platform, Enterprise Edition (J2EE) applications that use the WSPrincipalMappingLoginModule implementation directly in the Java Authentication and Authorization Service (JAAS) login when **Use Java 2 security to restrict application access to local resources** and the **Restrict access to resource authentication data** options are enabled.

Default: Disabled

Current realm definition:

Specifies the current setting for the active user repository.

This field is read-only.

Available realm definitions:

Specifies the available user account repositories.

Set as current:

Enables the user repository after it is configured.

LDAP or a custom user registry is required when running as a UNIX non-root user or running in a multi-node environment.

You can configure settings for one of the following user repositories:

Federated repositories

Specify this setting to manage profiles in multiple repositories under a single realm. The realm can consist of identities in:

- The file-based repository that is built into the system
- One or more external repositories
- Both the built-in, file-based repository and in one or more external repositories

Note: Only a user with administrator privileges can view the federated repositories configuration.

Local operating system

You cannot use localOS in multi-node or when running as non-root on a UNIX platform.

Standalone LDAP registry

Specify this setting to use standalone LDAP registry settings when users and groups reside in an external LDAP directory. When security is enabled and any of these properties change, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** or **OK** to validate the changes.

Note: Since multiple LDAP servers are supported, this setting does not imply one LDAP registry.

Standalone custom registry

Specify this setting to implement your own standalone custom registry that implements the com.ibm.websphere.security.UserRegistry interface. When security is enabled and any of these properties change, go to the Secure administration, applications, and infrastructure panel and click **Apply** or **OK** to validate the changes.

Default: Disabled

Use domain-qualified user names:

Specifies that user names that are returned by methods are qualified with the security domain in which they reside.

Default: Disabled

Active protocol:

Specifies the active authentication protocol for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI IIOP) requests, when security is enabled.

An Object Management Group (OMG) protocol called Common Secure Interoperability Version 2 (CSIv2) supports increased vendor interoperability and additional features. If all of the servers in your security domain are Version 5.x and later servers, specify CSI as your protocol.

V6.0.x If some servers are Version 3.x or Version 4.x servers, specify CSI and SAS.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Note: V6.0.x This field displays only when a Version 6.0.x and earlier server is detected in your environment.

Default: BOTH
Range: CSI and SAS, CSI

Specify extent of protection wizard settings:

Use this security wizard page to determine whether to enable application security and restrict access to local resources. When you use the wizard, admin security is enabled by default.

To view this security wizard page, click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.

Enable application security:

Enables application-level security unless the option is overwritten at the server level.

Default: Disabled

Use Java 2 security to restrict application access to local resources:

Specifies whether to enable or disable Java 2 security permission checking. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

When the **Use Java 2 security to restrict application access to local resources** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, the application might fail to run properly until the required permissions are granted in either the app.policy file or the was.policy file of the application. AccessControl exceptions are generated by applications that do not have all the required permissions. See “Java 2 security” on page 1061 for more information about Java 2 security.

Default: Disabled

Custom properties: Security:

Use this page to understand the predefined custom properties that are related to security.

To view this administrative console page, click **Security > Secure administration, applications, and infrastructure > Custom properties**. You can click **New** to add a new custom property and its associated value.

com.ibm.CSI.rmiInboundLoginConfig:

This property specifies the Java Authentication and Authorization Service (JAAS) login configuration that is used for Remote Method Invocation (RMI) requests that are received inbound.

By knowing the login configuration, you can plug in a custom login module that can handle specific cases for RMI logins.

Default system.RMI_INBOUND

com.ibm.CSI.rmiOutboundLoginConfig:

This property specifies the JAAS login configuration that is used for RMI requests that are sent outbound.

Primarily, this property prepares the propagated attributes in the Subject to be sent to the target server. However, you can plug in a custom login module to perform outbound mapping.

Default system.RMI_OUTBOUND

com.ibm.CSI.supportedTargetRealms:

This property enables credentials that are authenticated in the current realm to be sent to any realm that is specified in the Trusted target realms field. The Trusted target realms field is available on the CSIv2 outbound authentication panel. This property enables those realms to perform inbound mapping of the data from the current realm.

It is not recommended that you send authentication information to an unknown realm. Thus, this provides a way to specify that the alternate realms are trusted. To access the CSIv2 outbound authentication panel, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under RMI/IIOP security, click **CSIv2 outbound authentication**.

com.ibm.audit.auditPolicy:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default REQUIRED

com.ibm.audit.auditQueueSize:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default 5000

com.ibm.audit.auditServiceEnabled:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default false

com.ibm.audit.auditSpecification:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default J2EE=AUTHN=failure=enabled:J2EE=AUTHZ=failure=enabled

com.ibm.security.useFIPS:

Specifies that Federal Information Processing Standard (FIPS) algorithms are used. The application server uses the IBMJCEFIPS cryptographic provider instead of the IBMJCE cryptographic provider.

Default false

com.ibm.websphere.security.audit.auditEventFactory:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default J2EE=com.ibm.ws.security.audit.defaultAuditEventFactoryImpl

com.ibm.ws.security.defaultLoginConfig:

This property is the JAAS login configuration that is used for logins that do not fall under the WEB_INBOUND, RMI_OUTBOUND, or RMI_INBOUND login configuration categories.

Internal authentication and protocols that do not have specific JAAS plug points call the system login configuration that is referenced by com.ibm.ws.security.defaultLoginConfig configuration.

Default system.DEFAULT

com.ibm.ws.security.ssoInteropModeEnabled:

This property determines whether to send LtpaToken2 and LtpaToken cookies in the response to a Web request (interoperable).

When this property value is false, the application server just sends the new LtpaToken2 cookie which is stronger, but not interoperable with some other products and Application Server releases prior to Version 5.1.1. In most cases, the old LtpaToken cookie is not needed and you can set this property to false.

Default true

com.ibm.ws.security.webChallengeIfCustomSubjectNotFound:

This property determines the behavior of a single sign-on LtpaToken2 login.

When this property value is set to true, the token contains a custom cache key, and the custom Subject cannot be found, the token is used to log in directly as the custom information needs to be gathered again. A challenge occurs so that the user to login again. When this property value is set to false and the custom Subject is not found, the LtpaToken2 is used to login and gather all of the registry attributes. However, the token might not obtain any of the special attributes that downstream applications might expect.

Default true

com.ibm.ws.security.webInboundLoginConfig:

This property is the JAAS login configuration that is used for Web requests that are received inbound.

By knowing the login configuration, you can plug in a custom login module that can handle specific cases for Web logins.

Default system.WEB_INBOUND

com.ibm.ws.security.webInboundPropagationEnabled:

This property determines whether a received LtpaToken2 cookie should search for the propagated attributes locally before searching the original login server that is specified in the token. After the propagated attributes are received, the Subject is regenerated and the custom attributes are preserved.

You can configure the distributed replication service (DRS) to send the propagated attributes to front-end servers such that a local dynacache lookup can find the propagated attributes. Otherwise, an MBean request is sent to the original login server to retrieve these attributes.

Default true

com.ibm.wsspi.security.audit.auditServiceProvider:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default DEFAULT =
com.ibm.ws.security.audit.defaultAuditServiceProviderImpl

com.ibm.wsspi.security.ltpa.tokenFactory:

This property specifies the Lightweight Third Party Authentication (LTPA) token factories that can be used to validate the LTPA tokens.

Validation occurs in the order in which the token factories are specified because LTPA tokens do not have object identifiers (OIDs) that specify the token type. The Application Server validates the tokens using each token factory until validation is successful. The order that is specified for this property is the most likely order of the received tokens. Specify multiple token factories by separating them with a pipe (**|**) without spaces before or following the pipe.

Default com.ibm.ws.security.ltpa.LTPATokenFactory |
com.ibm.ws.security.ltpa.LTPAToken2Factory |
com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.authenticationTokenFactory:

This property specifies the implementation that is used for an authentication token in the attribute propagation framework. The property provides an old LTPA token implementation for use as the authentication token.

Default com.ibm.ws.security.ltpa.LTPATokenFactory

com.ibm.wsspi.security.token.authorizationTokenFactory:

This property specifies the implementation that is used for an authorization token. This token factory encodes the authorization information.

Default com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.propagationTokenFactory:

This property specifies the implementation that is used for a propagation token. This token factory encodes the propagation token information.

The propagation token is on the thread of execution and is not associated with any specific user Subjects. The token follows the invocation downstream wherever the process leads.

Default com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.singleSignonTokenFactory:

This property specifies the implementation that is used for a Single Sign-on (SSO) token. This implementation is the cookie that is set when propagation is enabled regardless of the state of the com.ibm.ws.security.ssoInteropModeEnabled property.

By default, this implementation is the LtpaToken2 cookie.

Default com.ibm.ws.security.ltpa.LTPAToken2Factory

security.enablePluggableAuthentication:

This property is no longer used. Instead, use WEB_INBOUND login configuration.

Complete the following steps to modify the WEB_INBOUND login configuration:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Java Authentication and Authorization Service, click **System logins**.

Default true

Security custom property collection:

Use this page to view and manage arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties.

The administrative console contains several Custom Properties pages that work similarly. To view one of these administrative pages, click a **Custom Properties** link.

Name:

Specifies the name (or key) for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property that has that name is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in the application server.

Value:

Specifies the value paired with the specified name.

Description:

Provides information about the name-value pair.

Security custom property settings:

Use this page to configure arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

The administrative console contains several Custom property settings pages that work similarly. To view one of these administrative pages, click **Custom Properties > New**.

Name:

Specifies the name (or key) for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property that has that name is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in WebSphere Application Server.

Data type String

Value:

Specifies the value paired with the specified name.

Data type String

Description:

Provides information about the name and value pair.

Data type String

Testing security after enabling it

Basic tests are available that show whether the fundamental security components are working properly. Use this task to validate your security configuration.

After configuring administrative security and restarting all of your servers in a secure mode, validate that security is properly enabled.

There are a few techniques that you can use to test the various security login types. For example, you can test the Web-based BasicAuth login, Web-based form login, and the Java client BasicAuth login.

Basic tests are available that show whether the fundamental security components are working properly. Complete the following steps to validate your security configuration:

1. After enabling security, verify that your system comes up in secure mode.
2. Test the Web-based BasicAuth with *Snoop*, by accessing the following URL: `http://hostname.domain:9080/snoop`. A login panel is displayed. If a login panel does not display, then a problem exists. If the panel appears, type in any valid user ID and password in your configured user registry.
3. Test the Web-based form login by starting the administrative console: `http://hostname.domain:port_number/ibm/console`. A form-based login page is displayed. If a login page does not appear, try accessing the administrative console by typing `https://myhost.domain:9043/ibm/console`.

   Type in the administrative user ID and password that are used for configuring your user registry when configuring security.
4. Test Test Java Client BasicAuth with *dumpNameSpace*.

   Use the `app_server_root/bin/dumpNameSpace.bat` file. A login panel appears. If a login panel does not appear, there is a problem. Type in any valid user ID and password in your configured user registry.
5. Test all of your applications in secure mode.
6. If all the tests pass, proceed with more rigorous testing of your secured applications. If you have any problems, review the output logs in the WebSphere Application Server `/logs/nodeagent` or WebSphere Application Server `/logs/server_name` directories, respectively. For more information on common problems, see Troubleshooting security configurations.

The results of these tests, if successful, indicate that security is fully enabled and working properly.

Authenticating users

The process of authenticating users involves a user registry and an authentication mechanism. Optionally, you can define trust between WebSphere Application Server and a proxy server, configure single sign-on capability, and specify how to propagate security attributes between application servers.

The following security topics are covered in this section:

User registries
For information on local operating system, Lightweight Directory Access Protocol (LDAP), custom user registries, and user repositories such as virtual member manager, see “User registries and repositories” on page 1084.

Authentication protocol for EJB security
For more information on the authentication protocols that are used for Enterprise JavaBeans (EJB) security, see “Authentication protocol for EJB security” on page 1228.

Trust associations
For more information on trust associations, see “Trust associations” on page 1194.

Single sign-on
For more information on single sign-on, see “Single sign-on” on page 1198.

Security attribute propagation
For more information on propagation tokens, authorization tokens, single sign-on tokens, and authentication tokens, see “Security attribute propagation” on page 1206.

The following information is covered in this section:

- Configure a user registry. For more information, see “Selecting a registry or repository.”
- Configure WebSEAL or a custom trust association interceptor. For more information see, “Integrating third-party HTTP reverse proxy servers” on page 1243.
- Configure single sign-on. For more information, see “Implementing single sign-on to minimize Web user authentications” on page 1245.
- Propagate security attributes. For more information, see “Propagating security attributes among application servers” on page 1283.
- Configure the authentication cache. For more information, see “Configuring the authentication cache” on page 1286.

After completing the configuring the authentication process, you must authorize access to resources. For more information, see “Authorizing access to resources” on page 1333.

Selecting a registry or repository

Information about users and groups reside in a user registry. In WebSphere Application Server, a user registry authenticates a user and retrieves information about users and groups to perform security-related functions, including authentication and authorization.

Note: During profile creation, either during installation or post-installation, administrative security is enabled by default. You might decide not to enable security, but if the default is accepted, the file-based federated user repository is configured as the active user registry. You can use a different user registry before the profile is created.

Before configuring the user registry or repository, decide which user registry or repository to use. Though different types of registries and repositories are supported, all of the processes in WebSphere Application Server can use only one active registry.

Configuring the correct registry or repository is a prerequisite to assigning users and groups to roles for applications. When a user registry or repository is not configured, the local operating system registry is used by default. If your choice of user registry is not the local operating system registry, you need to first configure the registry or repository, which is normally done as part of enabling security, restart the servers, and then assign users and groups to roles for all your applications.

In addition to local operating system and LDAP registries, WebSphere Application Server also provides a plug-in to support any registry by using the custom registry feature. The custom registry feature enables you to configure any user registry that is not made available through the security configuration panels of the WebSphere Application Server.

The UserRegistry interface is used to implement both the custom registry and the federated repository options for the user account repository. The interface is very helpful in situations where the current user and group information exists in some other formats, for example, a database, and cannot move to local operating system or LDAP registries. In such a case, you can implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all the security-related operations. The process of implementing a custom registry is a software implementation effort and it is expected that the implementation does not depend on other WebSphere Application Server resources, for example, data sources, for its operation.

WebSphere Application Server supports the following types of user registries:

- Federated repository
- Local operating system
- Standalone Lightweight Directory Access Protocol (LDAP) registry
- Standalone custom registry

After the applications are assigned users and groups and you need to change the user registries, delete all the users and groups, including any RunAs role, from the applications, and reassign them after changing the registry through the administrative console or by using wsadmin scripting. The following **wsadmin** command, which uses Jacl, removes all of the users and groups from any application:

```
$AdminApp deleteUserAndGroupEntries yourAppName
```

where *yourAppName* is the name of the application. Backing up the old application is advised before performing this operation. However, if both of the following conditions are true, you might be able to switch the registries without having to delete the users and groups information:

- All of the user and group names, including the password for the RunAs role users, in all of the applications match in both user registries.
- The application bindings file does not contain the access IDs which are unique for each user registry even for the same user or group name.

By default, an application does not contain access IDs in the bindings file. These IDs are generated when the applications start. However, if you migrated an existing application from an earlier release, or if you used the wsadmin script to add access IDs for the applications to improve performance, you have to remove the existing user and group information and add the information after configuring the new user registry.

For more information on updating access IDs, see updateAccess IDs in the AdminApp object for scripted administration article.

Complete one of the following steps to configure your user registry:

- “Configuring local operating system registries” on page 1085
- “Configuring Lightweight Directory Access Protocol user registries” on page 1088
- “Configuring standalone custom registries” on page 1107.
- “Managing the realm in a federated repository configuration” on page 1129

1. If you are enabling security, make sure that you complete the remaining steps. Verify that the User account repository on the Secure administration, applications, and infrastructure panel is set to the appropriate registry or repository. As the final step, validate the user ID and the password by clicking **Apply** on the Secure administration, applications, and infrastructure panel. Save, stop and start all WebSphere Application Servers.
2. For any changes in user registry panels to be effective, you must validate the changes by clicking **Apply** on the Secure administration, applications, and infrastructure panel. After validation, save the configuration and stop and start all WebSphere Application Servers, including the cells, nodes and all of the application servers. To avoid inconsistencies between the WebSphere Application Server processes, make sure that any changes to the registry or repository are done when all of the processes are running. If any of the processes are down, force synchronization to make sure that the process can start later.

   If the server or servers start without any problems, the setup is correct.

User registries and repositories

Information about users and groups reside within a registry or repository.

WebSphere Application Server provides implementations that support multiple types of registries and repositories including the local operating system registry, a standalone Lightweight Directory Access Protocol (LDAP) registry, a standalone custom registry, and federated repositories.

With WebSphere Application Server, a user registry or a repository, such as virtual member manager, authenticates a user and retrieves information about users and groups to perform security-related functions including authentication and authorization.

With WebSphere Application Server, a user registry or repository is used for:

- Authenticating a user using basic authentication, identity assertion, or client certificates
- Retrieving information about users and groups to perform security-related administrative functions, such as mapping users and groups to security roles

Although WebSphere Application Server supports different types of user registries, only one user registry can be active. This active registry is shared by all of the product server processes.

After configuring the registry or repository, you must specify it as the active repository. Through the administration console, you can select an available realm definition for the registry or repository from the User account repository section of the Secure administration, applications, and administration panel. After selecting the registry or repository, first click **Set as current**, and then click **Apply**.

Note: WebSphere Application Server has implemented a user registry proxy by using the UserRegistry interface. However, the return values are little different from the interface. For example, getUniqueUserId returns the uniqueID with the realm name wrapped. You cannot use the return value to pass to getUserSecurityName, as shown in the following example:

```
// Retrieves the default InitialContext for this server.
javax.naming.InitialContext ctx = new javax.naming.InitialContext();

// Retrieves the local UserRegistry object.
com.ibm.websphere.security.UserRegistry reg =
         (com.ibm.websphere.security.UserRegistry) ctx.lookup("UserRegistry");

// Retrieves the registry uniqueID based on the userName that is specified
     // in the NameCallback.
String uniqueid = reg.getUniqueUserId(userName);
// Strip the realm name and get real uniqueID
String uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);

// Retrieves the security name from the user registry based on the uniqueID.
String securityName = reg.getUserSecurityName(uid);
```

Configuring local operating system registries

Use these steps to configure local operating system registries.

For security purposes, the WebSphere Application Server provides and supports the implementation for Windows operating system registries, AIX, Solaris and multiple versions of Linux operating systems. The respective operating system application programming interface (API) are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). Access to these APIs are restricted to users who have special privileges. These privileges depend on the operating system and are described below.

In WebSphere Application Server Version 6.1, you can use an internally-generated server ID because the Security WebSphere Common Configuration Model (WCCM) model contains a new tag, internalServerId. You do not need to specify a server user ID and a password during security configuration except in a mixed-cell environment. See “Administrative roles and naming service authorization” on page 1334 for more detailed information about the new internal server ID.

Windows Consider the following issues:

- The server ID needs to be different from the Windows machine name where the product is installed. For example, if the Windows machine name is `vicky` and the security server ID is `vickyy`, the Windows system fails when getting the information (group information, for example) for user `vicky`.
- WebSphere Application Server dynamically determines whether the machine is a member of a Windows system domain.
- WebSphere Application Server does not support Windows trusted domains.
- If a machine is a member of a Windows domain, both the domain user registry and the local user registry of the machine participate in authentication and security role mapping.

- The domain user registry takes precedence over the local user registry of the machine and can have undesirable implications if users with the same password exist in both user registries.
- The user that the product processes run under requires the `Administrative` and `Act as part of the operating system` privileges to call the Windows operating system APIs that authenticate or collect user and group information. The process needs special authority, which is given by these privileges. The user in this example might not be the same as the security server ID (the requirement for which is a valid user in the registry). This user logs into the machine (if using the command line to start the product process) or the Log On User setting in the services panel if the product processes have started using the services. If the machine is also part of a domain, this user is a part of the Domain Admin group in the domain to call the operating system APIs in the domain in addition to having the Act as part of operating system privilege in the local machine.

Consider the following points:

- AIX Solaris The user that the product processes run under requires the `root` privilege. This privilege is needed to call the operating system APIs to authenticate or to collect user and group information. The process needs special authority, which is given by the `root` privilege. This user might not be the same as the security server ID (the requirement is that it should be a valid user in the registry). This user logs into the machine and is running the product processes.
- Linux You might need to have the password shadow file in your system.

Important: The local operating system in not a valid user account repository when you have a mixed cell environment that includes both z/OS platform and non-z/OS platform nodes.

The following steps are needed to perform this task initially when setting up security for the first time.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Local operating system** and click **Configure**.
3. Enter a valid user name in the **Primary administrative user name** field. This value is the name of a user with administrative privileges that is defined in the registry. This user name is used to access the administrative console or used by wsadmin.
4. Click **Apply**.
5. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

   Server user ID or administrative user on a Version 6.0.x node
   Specify the short name of the account that is chosen in the second step.

   Server user password
   Specify the password of the account that is chosen in the second step.
6. Click **OK**.

   The administrative console does not validate the user ID and password when you click **OK**. Validation is only done when you click **OK** or **Apply** in the Secure administration, applications, and infrastructure panel. First, make sure that you select **Local operating system** as the available realm definition in the User account repository section, and click **Set as current**. If security was already enabled and you had changed either the user or the password information in this panel, make sure to go to the Secure administration, applications, and infrastructure panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

   Important: Until you authorize other users to perform administrative functions, you can only access the administrative console with the server user ID and password that you specified. For more information, see “Authorizing access to administrative roles” on page 1386.

For any changes in this panel to be effective, you need to save, stop, and start all the product servers, including deployment managers, nodes and application servers. If the server comes up without any problems, the setup is correct.

After completed these steps, you have configured WebSphere Application Server to use the local operating system registry to identify authorized users.

Complete any remaining steps for enabling security. For more information, see “Enabling security” on page 1047.

Configuring user ID for proper privileges:

Use this page to configure a user ID for proper privileges or to log on as a service on the Windows platform.

Windows

1. Click **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Local Security Policy** > **Local Policies** > **User Rights Assignments** > **Act as part of the operating system** (or **Log on as a service**). For a Windows domain controller, replace **Local Security Policy** with **Domain Security Policy** in the first step.

   Note: If the machine is a standalone machine and not a member of a domain, you must add a machineName*userID*, where the *userID* is the owner of the process, such as WebSphere Application Server. If you run WebSphere Application Server as a service, you can log on with `localsystem` as the service.
2. If the machine is a member of a domain, add domainName*userID*, where the *userID* is the owner of process (such as WebSphere Application Server). Start WebSphere Application Server as a service with login ID domainName\userID. If WebSphere Application Server is already in service, go to the service and right-click **IBM WebSphere Application Server > properties >Logon to change the logon ID and password** to restart WebSphere Application Server.
3. Add the user name by clicking **Add**.
4. Restart the machine.

Note: In all of the previous configurations, the server can be run as a service using `LocalSystem` for the Log On As entry. The `LocalSystem` entry has the required privileges and there is no need to give special privileges to any user. However, because the `LocalSystem` entry has special privileges, make sure that it is appropriate to use in your environment.

Local operating system settings:

Use this page to configure local operating system registry settings.

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

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Local operating system**.
3. Click **Configure**.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are adding a Version 5.0.*x* or 6.0.*x* node to a Version 6.1 cell, you must ensure that the Version 5.*x* or Version 6.0.*x* server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your local operating system.

The user name is used to log on to the administrative console when administrative security is enabled.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Local operating system wizard settings:

Use this security wizard page to configure local operating system registry settings.

To view this security wizard page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Local operating system** option and click **Next**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your local operating system.

The user name is used to log on to the administrative console when administrative security is enabled.

Configuring Lightweight Directory Access Protocol user registries

To access a user registry using the Lightweight Directory Access Protocol (LDAP), you must know a valid user name (ID) and password, the server host and port of the registry server, the base distinguished name (DN) and, if necessary, the bind DN and the bind password. You can choose any valid user in the user registry that is searchable. You can use any user ID that has the administrative role to log in.

In some LDAP servers, administrative users cannot be searched and thus cannot be used, for example, when `cn=root` in Tivoli Access Manager. The user is referred to as a WebSphere Application Server security server ID, server ID, or server user ID in the documentation. A server ID user has special privileges when calling some protected internal methods.

Normally, the primary administrative user name is used to log into the administrative console if security is enabled. By default, security is enabled after installation.

When security is enabled in the product, the primary administrative user name and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. It is important to choose an ID and password that do not expire or change often. If the product server user ID or password need to change in the registry, make sure that the changes are performed when all the product servers are up and running. When changes are to be made in the registry, review the article on "Standalone Lightweight Directory Access Protocol registries" on page 1183 (LDAP) before beginning this task.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Enter a valid user name in the **Primary administrative user name** field. You can either enter the complete distinguished name (DN) of the user or the short name of the user, as defined by the user filter in the Advanced LDAP settings panel. For example, enter the user ID for Netscape browsers. This ID is the security server ID, which is only used for WebSphere Application Server security and is not associated with the system process that runs the server. The server calls the local operating system registry to authenticate and obtain privilege information about users by calling the native application programming interfaces (API) in that particular registry.
4. **Optional:** If you want to use the server ID that is stored in the repository, complete the following:
   a. Select **Automatically generated server identity** to enable the application server to generate the server identity that is used for internal process communication. You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the **Internal server ID** field.
   b. Alternatively, specify a user identity in the repository that is used for internal process communication in the **Server identity that is stored in the repository** field.
   c. Alternatively, specify the user ID that is used to run the application server for security purposes in the **Server user ID or administrative user on a Version 6.0.x node** field.
5. Select the type of LDAP server to use from the **Type** list. The type of LDAP server determines the default filters that are used by WebSphere Application Server. These default filters change the **Type** field to **Custom**, which indicates that custom filters are used. This action occurs after you click **OK** or **Apply** in the Advanced LDAP settings panel. Choose the **Custom** type from the list and modify the user and group filters to use other LDAP servers, if required.

   IBM Tivoli Directory Server users can choose IBM Tivoli Directory Server as the directory type. Use the IBM Tivoli Directory Server directory type for better performance. For a list of supported LDAP servers, see the Supported hardware, software, and APIs Web site.

   Attention: IBM SecureWay Directory Server has been renamed to IBM Tivoli Directory Server in WebSphere Application Server version 6.1.
6. Enter the fully qualified host name of the LDAP server in the **Host** field. You can enter either the IP address or domain name system (DNS) name.
7. Enter the LDAP server port number in the **Port** field. The host name and the port number represent the realm for this LDAP server in the WebSphere Application Server cell. So, if servers in different cells are communicating with each other using Lightweight Third Party Authentication (LTPA) tokens, these realms must match exactly in all the cells.

   The default value is 389. If multiple WebSphere Application Servers are installed and configured to run in the same single sign-on domain, or if the WebSphere Application Server interoperates with a previous version of the WebSphere Application Server, then it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as 389 in a version 5.*x*

configuration, and a WebSphere Application Server at version 6.0.*x* is going to interoperate with the version 5.*x* server, then verify that port 389 is specified explicitly for the version 6.0.*x* server.

8. Enter the base distinguished name (DN) in the **Base distinguished name** field. The base DN indicates the starting point for searches in this LDAP directory server. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, specify the base DN as any of the following options assuming a suffix of `c=us`): `ou=Rochester, o=IBM, c=us` or `o=IBM c=us` or `c=us`. For authorization purposes, this field is case sensitive by default. Match the case in your directory server. If a token is received (for example, from another cell or Lotus Domino) the base DN in the server must match exactly the base DN from the other cell or Domino. If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option.

   In WebSphere Application Server, the distinguished name is normalized according to the Lightweight Directory Access Protocol (LDAP) specification. Normalization consists of removing spaces in the base distinguished name before or after commas and equal symbols. An example of a non-normalized base distinguished name is `o = ibm, c = us` or `o=ibm, c=us`. An example of a normalized base distinguished name is `o=ibm,c=us`.

   To interoperate between WebSphere Application Server Version 5 and later versions, you must enter a normalized base distinguished name in the **Base Distinguished Name** field. In WebSphere Application Server, Version 5.0.1 or later, the normalization occurs automatically during runtime.

   This field is required for all LDAP directories except the Lotus Domino Directory. The **Base Distinguished Name** field is optional for the Domino server.

9. **Optional:** Enter the bind DN name in the **Bind distinguished name** field. The bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information. If the LDAP server is set up to use anonymous binds, leave this field blank. If a name is not specified, the application server binds anonymously. See the **Base Distinguished Name** field description for examples of distinguished names.
10. **Optional:** Enter the password corresponding to the bind DN in the **Bind password** field.
11. **Optional:** Modify the Search time out value. This timeout value is the maximum amount of time that the LDAP server waits to send a response to the product client before stopping the request. The default is 120 seconds.
12. Ensure that the **Reuse connection** option is selected. This option specifies that the server should reuse the LDAP connection. Clear this option only in rare situations where a router is used to send requests to multiple LDAP servers and when the router does not support affinity. Leave this option selected for all other situations.
13. **Optional:** Verify that the **Ignore case for authorization** option is enabled. When you enable this option, the authorization check is case insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the LDAP server and is case sensitive. However, when you use either the IBM Directory Server or the Sun ONE (formerly iPlanet) Directory Server LDAP servers, you must enable this option because the group information that is obtained from the LDAP servers is not consistent in case. This inconsistency affects the authorization check only. Otherwise, this field is optional and can be enabled when a case sensitive authorization check is required. For example, you might select this option when you use certificates and the certificate contents do not match the case of the entry in the LDAP server.

    You can also enable the **Ignore case for authorization** option when you are using single sign-on (SSO) between the product and Lotus Domino. The default is enabled.
14. **Optional:** Select the **SSL enabled** option if you want to use Secure Sockets Layer communications with the LDAP server.

    If you select the **SSL enabled** option, you can select either the **Centrally managed** or the **Use specific SSL alias** option.

    Centrally managed

    Enables you to specify an SSL configuration for particular scope such as the cell, node, server, or cluster in one location. To use the **Centrally managed** option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security

configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an LDAP registry, you can override the inherited SSL configuration by specifying an SSL configuration for LDAP. To specify an SSL configuration for LDAP, complete the following steps:

a. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.

b. Expand **Outbound >** *cell_name* **> Nodes >** *node_name* **> Servers >** *server_name* **> LDAP**.

Use specific SSL alias

Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu below the option.

This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the following steps:

a. Click **Security > SSL certificate and key management**.

b. Under Configuration settings, click **Manage endpoint security configurations**.

c. Select a Secure Sockets Layer (SSL) *configuration_name* for selected scopes, such as a cell, node, server, or cluster.

d. Under Related items, click **SSL configurations**.

e. Click **New**.

15. Click **OK** and either **Apply** or **Save** until you return to the Secure administration, applications, and infrastructure panel.

This set of steps is required to set up the LDAP user registry. This step is required as part of enabling security in the WebSphere Application Server.

1. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071.
2. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems the setup is correct.

Standalone LDAP registry settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) settings when users and groups reside in an external LDAP directory.

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

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.

When security is enabled and any of these properties change, go to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are adding a Version 5.0.*x* or 6.0.*x* node to a Version 6.1 cell, you must ensure that the Version 5.*x* or Version 6.0.*x* server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Type of LDAP server:

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is not supported.

Host:

Specifies the host ID (IP address or domain name service (DNS) name) of the LDAP server.

Port:

Specifies the host port of the LDAP server.

If multiple application servers are installed and configured to run in the same single sign-on domain or if the application server interoperates with a previous version, it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as 389 in a Version 4.0.x configuration, and a WebSphere Application Server at Version 5 is going to interoperate with the Version 4.0.x server, verify that port 389 is specified explicitly for the Version 5 server.

Default: 389
Type: Integer

Base distinguished name (DN):

Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

For example, for a user with a DN of `cn=John Doe , ou=Rochester, o=IBM, c=US`, specify the Base DN as any of the following options: `ou=Rochester, o=IBM, c=US` or `o=IBM c=US` or `c=US`. For authorization purposes, this field is case sensitive. This specification implies that if a token is received, for example, from another cell or Lotus Domino, the base DN in the server must match the base DN from the other cell or Lotus Domino server exactly. If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option. This option is required for all Lightweight Directory Access Protocol (LDAP) directories, except for the Lotus Domino Directory, IBM Tivoli Directory Server V6.0, and Novell eDirectory, where this field is optional.

If you need to interoperate between the application server Version 5 and a Version 5.0.1 or later server, you must enter a normalized base DN. A normalized base DN does not contain spaces before or after commas and equal symbols. An example of a non-normalized base DN is `o = ibm, c = us` or `o=ibm, c=us`. An example of a normalized base DN is `o=ibm,c=us`. In WebSphere Application Server, Version 5.0.1 or later, the normalization occurs automatically during runtime.

Bind distinguished name (DN):

Specifies the DN for the application server to use when binding to the directory service.

If no name is specified, the application server binds anonymously. See the Base distinguished name (DN) field description for examples of distinguished names.

Bind password:

Specifies the password for the application server to use when binding to the directory service.

Search timeout:

Specifies the timeout value in seconds for an Lightweight Directory Access Protocol (LDAP) server to respond before stopping a request.

Default: 120

Reuse connection:

Specifies whether the server reuses the LDAP connection. Clear this option only in rare situations where a router is used to distribute requests to multiple LDAP servers and when the router does not support affinity.

Default: Enabled
Range: Enabled or Disabled

Important: Disabling the **Reuse connection** option causes the application server to create a new LDAP connection for every LDAP search request. This situation impacts system performance if your environment requires extensive LDAP calls. This option is provided because the router is not sending the request to the same LDAP server. The option is also used when the idle connection timeout value or firewall timeout value between the application server and LDAP is too small.

If you are using WebSphere Edge Server for LDAP failover, you must enable TCP resets with the Edge server. A TCP reset causes the connection to immediately closed and a backup server to failover. For more information, see ″Sending TCP resets when server is down″ at http://www-3.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/edge/LBguide.htm#HDRRESETSERVER and the Edge Server V2 - TCP Reset feature in PTF #2 described in: ftp://ftp.software.ibm.com/software/websphere/edgeserver/info/doc/v20/en/updates.pdf.

Ignore case for authorization:

Specifies that a case insensitive authorization check is performed when using the default authorization.

This option is required when IBM Tivoli Directory Server is selected as the LDAP directory server.

This option is required when Sun ONE Directory Server is selected as the LDAP directory server. For more information, see ″Using specific directory servers as the LDAP server″ in the documentation.

This option is optional and can be enabled when a case-sensitive authorization check is required. For example, use this option when the certificates and the certificate contents do not match the case that is used for the entry in the LDAP server. You can enable the **Ignore case for authorization** option when using single sign-on (SSO) between the application server and Lotus Domino.

Default: Enabled
Range: Enabled or Disabled

SSL enabled:

Specifies whether secure socket communication is enabled to the Lightweight Directory Access Protocol (LDAP) server.

When enabled, the LDAP Secure Sockets Layer (SSL) settings are used, if specified.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Default: Enabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

Standalone LDAP registry wizard settings:

Use this security wizard page to provide the basic settings to connect the application server to an existing Lightweight Directory Access Protocol (LDAP) registry.

To view this security wizard page, click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**. You can modify your LDAP registry configuration by completing the following steps:

1. Click **Security > Security administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

Type of LDAP server:

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is not supported.

Host:

Specifies the host ID (IP address or domain name service (DNS) name) of the LDAP server.

Port:

Specifies the host port of the LDAP server.

If multiple application servers are installed and configured to run in the same single sign-on domain or if the application server interoperates with a previous version, it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as `389` in a Version 4.0.x configuration, and a WebSphere Application Server at Version 5 is going to interoperate with the Version 4.0.x server, verify that port `389` is specified explicitly for the Version 5 server.

Default:	389
Type:	Integer

Base distinguished name (DN):

Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

For example, for a user with a DN of `cn=John Doe , ou=Rochester, o=IBM, c=US`, specify the Base DN as any of the following options: `ou=Rochester, o=IBM, c=US` or `o=IBM, c=US` or `c=US`. For authorization purposes, this field is case sensitive. This specification implies that if a token is received, for example, from another cell or Lotus Domino, the base DN in the server must match the base DN from the other cell or Lotus Domino server exactly.

If you need to interoperate between the application server Version 5 and a Version 5.0.1 or later server, you must enter a normalized base DN. A normalized base DN does not contain spaces before or after commas and equal symbols. An example of a non-normalized base DN is `o = ibm, c = us` or `o=ibm, c=us`. An example of a normalized base DN is `o=ibm,c=us`. In WebSphere Application Server, Version 5.0.1 or later, the normalization occurs automatically during run time.

Bind distinguished name (DN):

Specifies the DN for the application server to use when binding to the directory service.

If no name is specified, the application server binds anonymously. See the Base distinguished name (DN) field description for examples of distinguished names.

Bind password:

Specifies the password for the application server to use when binding to the directory service.

Advanced Lightweight Directory Access Protocol user registry settings:

Use this page to configure the advanced Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups reside in an external LDAP directory.

To view this administrative page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

Default values for all the user and group related filters are already completed in the appropriate fields. You can change these values depending on your requirements. These default values are based on the type of LDAP server that is selected in the Standalone LDAP registry settings panel. If this type changes, for example from Netscape to Secureway, the default filters automatically change. When the default filter values change, the LDAP server type changes to `Custom` to indicate that custom filters are used. When security is enabled and any of these properties change, go to the Secure administration, applications, and infrastructure panel and click **Apply** or **OK** to validate the changes.

User filter:

Specifies the LDAP user filter that searches the user registry for users.

This option is typically used for security role-to-user assignments and specifies the property by which to look up users in the directory service. For example, to look up users based on their user IDs, specify `(&(uid=%v)(objectclass=inetOrgPerson))`. For more information about this syntax, see the LDAP directory service documentation.

Data type: String

Group filter:

Specifies the LDAP group filter that searches the user registry for groups

This option is typically used for security role-to-group assignments and specifies the property by which to look up groups in the directory service. For more information about this syntax, see the LDAP directory service documentation.

Data type: String

User ID map:

Specifies the LDAP filter that maps the short name of a user to an LDAP entry.

Specifies the piece of information that represents users when users display. For example, to display entries of the `object class = inetOrgPerson` type by their IDs, specify `inetOrgPerson:uid`. This field takes multiple objectclass:property pairs delimited by a semicolon (;).

Data type: String

Group ID map:

Specifies the LDAP filter that maps the short name of a group to an LDAP entry.

Specifies the piece of information that represents groups when groups display. For example, to display groups by their names, specify `*:cn`. The asterisk (*) is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs, delimited by a semicolon (;).

Data type: String

Group member ID map:

Specifies the LDAP filter that identifies user-to-group relationships.

For directory types SecureWay, and Domino, this field takes multiple objectclass:property pairs, delimited by a semicolon (;). In an objectclass:property pair, the object class value is the same object class that is defined in the group filter, and the property is the member attribute. If the object class value does not match the object class in the group filter, authorization might fail if groups are mapped to security roles. For more information about this syntax, see your LDAP directory service documentation.

For IBM Directory Server, Sun ONE, and Active Directory, this field takes multiple `group attribute:member attribute` pairs delimited by a semicolon (;). These pairs are used to find the group memberships of a user by enumerating all the group attributes that are possessed by a given user. For example, attribute pair `memberof:member` is used by Active Directory, and ibm-allGroup:member is used by IBM Directory Server. This field also specifies which property of an object class stores the list of members belonging to the group represented by the object class. For supported LDAP directory servers, see ″Supported directory services″.

Data type: String

Perform a nested group search:

Specifies a recursive nested group search.

Select this option if the Lightweight Directory Access Protocol (LDAP) server does not support recursive server-side group member searches and if recursive group member search is required. It is not recommended that you select this option to locate recursive group memberships for LDAP servers. Application server security leverages the recursive search functionality of the LDAP server to search a user's group memberships, including recursive group memberships. For example:

- IBM Directory Server is preconfigured by the application server security to recursively calculate a user's group memberships using the `ibm-allGroup` attribute.
- SunONE directory server is preconfigured to calculate nested group memberships using the `nsRole` attribute.

Data type: String

Certificate map mode:

Specifies whether to map X.509 certificates into an LDAP directory by EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.

Data type: String

Certificate filter:

Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP registry.

If more than one LDAP entry matches the filter specification at runtime, authentication fails because the result is an ambiguous match. The syntax or structure of this filter is: `(&(uid=${SubjectCN})(objectclass=inetOrgPerson))`. The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- `${UniqueKey}`
- `${PublicKey}`
- `${Issuer}`
- `${NotAfter}`
- `${NotBefore}`
- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectCN}`
- `${Version}`

Data type: String

Configuring Lightweight Directory Access Protocol search filters:

Use this topic to configure the LDAP search filters. These steps are required to modify existing user and group filters for a particular LDAP directory type, and also to set up certificate filters to map certificates to entries in the LDAP server.

WebSphere Application Server uses Lightweight Directory Access Protocol (LDAP) filters to search and obtain information about users and groups from an LDAP directory server. A default set of filters is provided for each LDAP server that the product supports. You can modify these filters to fit your LDAP configuration. After the filters are modified and you click **OK** or **Apply** the directory type in the Standalone LDAP registry panel changes to `custom`, which indicates that custom filters are used. Also, you can develop filters to support any additional type of LDAP server. The effort to support additional LDAP directories is optional and other LDAP directory types are not supported. Complete the following steps in the administrative console.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

4. Modify the user filter, if necessary. The user filter is used for searching the registry for users and is typically used for the security role-to-user assignment. The filter is also used to authenticate a user with the attribute that is specified in the filter. The filter specifies the property that is used to look up users in the directory service.

   In the following example, the property that is assigned to `%v`, which is the short name of the user, must be a unique key. Two LDAP entries with the same object class cannot have the same short name. To look up users based on their user IDs (uid) and to use the inetOrgPerson object class, specify the following syntax:

   ```
   (&(uid=%v)(objectclass=inetOrgPerson)
   ```

   For more information about this syntax, see the "Using specific directory servers as the LDAP server" on page 1102 documentation.

5. Modify the group filter, if necessary. The group filter is used in searching the registry for groups and is typically used for the security role-to-group assignment. Also, the filter is used to specify the property by which to look up groups in the directory service.

   In the following example, the property that is assigned to `%v`, which is the short name of the group, must be a unique key. Two LDAP entries with the same object class cannot have the same short name. To look up groups based on their common names (CN) and to use either the groupOfNames object class or the groupOfUniqueNames object class, specify the following syntax:

   ```
   (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
   ```

   For more information about this syntax, see the "Using specific directory servers as the LDAP server" on page 1102 documentation.

6. Modify the user ID map, if necessary. This filter maps the short name of a user to an LDAP entry and specifies the piece of information that represents users when these users are displayed with their short names. For example, to display entries of `object class = inetOrgPerson` by their IDs, specify `inetOrgPerson:uid`. This field takes multiple objectclass:property pairs, delimited by a semicolon (;). To provide a consistent value for methods like the getCallerPrincipal method and the getUserPrincipal method, the short name that is obtained by using this filter is used. For example, the `CN=Bob Smith, ou=austin.ibm.com, o=IBM, c=US` user can log in using any attributes that are defined, for example, e-mail address, social security number, and so on, but when these methods are called, the `bob` user ID is returned no matter how the user logs in.

7. Modify the group ID map filter, if necessary. This filter maps the short name of a group to an LDAP entry and specifies the piece of information that represents groups when groups display. For example, to display groups by their names, specify `*:cn`. The asterisk (*) is a wildcard character that searches on any object class in this case. This field takes multiple objectclass:property pairs, delimited by a semicolon (;).

8. Modify the group member ID map filter, if necessary. This filter identifies user-to-group memberships. For SecureWay, and Domino directory types, this field is used to query all the groups that match the specified object classes to see if the user is contained in the specified attribute. For example, to get all the users that belong to groups with the groupOfNames object class and the users that are contained in the member attributes, specify `groupOfNames:member`. This syntax, which is a property of an object class, stores the list of members that belong to the group that is represented by the object class. This field takes multiple objectclass:property pairs that are delimited by a semicolon (;). For more information about this syntax, see the "Using specific directory servers as the LDAP server" on page 1102.

   For the IBM Tivoli Directory Server, Sun ONE, and Active Directory, this field is used to query all users in a group with the information that is stored in the user object. For example, the memberof:member filter (for Active Directory) is used to get the memberof attribute of the user object to obtain all the groups to which the user belongs. The member attribute is used to get all the users in a group that use the Group object. Using the User object to obtain the group information improves performance.

9. Select the **Perform a nested group search** option if your LDAP server does not support recursive server-side searches.
10. Modify the Certificate map mode, if necessary. You can use the X.590 certificates for user authentication when LDAP is selected as the registry. This field is used to indicate whether to map the X.509 certificates into an LDAP directory user by **EXACT_DN** or **CERTIFICATE_FILTER**. If **EXACT_DN** is selected, the DN in the certificate must exactly match the user entry in the LDAP server, including case and spaces.

    Select the **Ignore case for authorization** option on the Standalone LDAP registry settings to make the authorization case insensitive. To access the Standalone LDAP registry settings panel, complete the following steps:

    a. Click **Security > Secure administration, applications, and infrastructure**.

    b. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**.
11. If you select **CERTIFICATE_FILTER**, specify the LDAP filter for mapping attributes in the client certificate to entries in LDAP. If more than one LDAP entry matches the filter specification at run time, authentication fails because an ambiguous match results. The syntax or structure of this filter is: `LDAP attribute=${Client certificate attribute}` (for example, `uid=${SubjectCN}`).

    The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. Note that the right side must begin with a dollar sign ($), open bracket ({), and end with a close bracket (}). Use the following certificate attribute values on the right side of the filter specification. The case of the strings is important.
    - ${UniqueKey}
    - ${PublicKey}
    - ${Issuer}
    - ${NotAfter}
    - ${NotBefore}
    - ${SerialNumber}
    - ${SigAlgName}
    - ${SigAlgOID}
    - ${SigAlgParams}
    - ${SubjectDN}
    - ${Version}

    To enable this field, select **CERTIFICATE_FILTER** for the certificate mapping.
12. Click **Apply**.

    When any LDAP user or group filter is modified in the Advanced LDAP Settings panel click **Apply**. Clicking **OK** navigates you to the Standalone LDAP registry panel, which contains the previous LDAP directory type, rather than the custom LDAP directory type. Clicking **OK** or **Apply** in the Standalone LDAP registry panel saves the back-level LDAP directory type and the default filters of that directory. This action overwrites any changes to the filters that you made. To avoid overwriting changes, you can take either of the following actions:
    - Click **Apply** in the Advanced Lightweight Directory Access Protocol (LDAP) user registry settings panel. Click **Security > Secure administration, applications, and infrastructure** and change the User account repository type to `Standalone custom registry`.
    - Select **Custom** type from the Standalone LDAP registry panel. Click **Apply** and then change the filters by clicking the Advanced Lightweight Directory Access Protocol (LDAP) user registry settings panel. After you complete your changes, click **Apply** or **OK**.

    The validation of the changes does not take place in this panel. Validation is done when you click **OK** or **Apply** on the Secure administration, applications, and infrastructure panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Secure administration, applications, and infrastructure panel. Select **Standalone LDAP registry** as the user account repository. If security is already enabled and any information on this panel changes, go to the

Secure administration, applications, and infrastructure panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

These steps result in the configuration of the LDAP search filters. These steps are required to modify existing user and group filters for a particular LDAP directory type. The steps are also used to set up certificate filters to map certificates to entries in the LDAP server.

1. Validate this setup by clicking **OK** or **Apply** on the Secure administration, applications, and infrastructure panel.
2. Save, stop, and start all the product servers, including the cell, nodes and all of the application servers for any changes in this panel to become effective.
3. After the server starts, go through all the security-related tasks (getting users, getting groups, and so on) to verify that the changes to the filters function.

Updating LDAP binding information:

Use this information to dynamically update security LDAP binding information by switching to a different binding identity.

You can dynamically update Lightweight Directory Access Protocol (LDAP) binding information without first stopping and restarting WebSphere Application Server by using the **wsadmin** tool.

The resetLdapBindInfo method in SecurityAdmin MBean is used to dynamically update LDAP binding information at WebSphere Application Server security runtime, and it takes the bind distinguished name (DN) and bind password parameters as input. The resetLdapBindInfo method validates the bind information against the LDAP server. If validation passes, new binding information is stored in `security.xml`, and a copy of the information is placed in WebSphere Application Server security runtime.

If the new binding information is `null`, `null`, the resetLdapBindInfo method first extracts LDAP binding information, including bind DN, bind password, and target binding host from WebSphere Application Server security configuration in `security.xml`. It then pushes the binding information to WebSphere Application Server security runtime.

There are two ways to dynamically update WebSphere Application Server security LDAP binding information using the SecurityAdmin MBean through wsadmin:

- "Switching to a different binding identity"
- "Switching to a failover LDAP host" on page 1102

Switching to a different binding identity:

To dynamically update security LDAP binding information by switching to a different binding identity:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.
3. Create a new bind DN. It must have the same access authority as the current bind DN.
4. Run the SecurityAdmin MBean across all of the application server processes to validate the new binding information, to save it to `security.xml`, and to push the new binding information to the runtime.

The following is a sample Jacl file for step 4:

```
proc LDAPReBind {args} {
  global AdminConfig AdminControl ldapBindDn ldapBindPassword
  set ldapBindDn [lindex $args 0]
  set ldapBindPassword [lindex $args 1]
       set secMBeans [$AdminControl queryNames type=SecurityAdmin,*]
       set plist  [list $ldapBindDn $ldapBindPassword]
```

```
        foreach secMBean $secMBeans {
              set result [$AdminControl invoke $secMBean resetLdapBindInfo $plist]
        }
}
```

Switching to a failover LDAP host:

To dynamically update security LDAP binding information by switching to a failover LDAP host:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
3. Change the password for bind DN on one LDAP server (it can be the primary or the backup).
4. Update the new bind DN password to WebSphere Application security runtime by calling resetLdapBindInfo with the bind DN and by using its new password as a parameter.
5. Use the new bind DN password for all of the other LDAP servers. The binding information is now consistent across WebSphere Application Server and the LDAP servers.

   Note: If you call resetLdapBindInfo with `null`, `null` as input parameters, WebSphere Application Server security runtime completes the following steps:

   a. Reads the bind DN, bind password, and target LDAP hosts from `security.xml`.

   b. Refreshes the cached connection to the LDAP server.

   If you configure security to use multiple LDAP servers, this MBean call forces WebSphere Application Server security to reconnect to the first available LDAP host in the list. For example, if three LDAP servers are configured in the order of L1, L2, and L3, the reconnection process always starts with the L1 server.

Using specific directory servers as the LDAP server:

This article provides important information about the directory servers that are supported as Lightweight Directory Access Protocol (LDAP) servers in WebSphere Application Server.

Microsoft Active Directory forest is not supported in the user registry in this product.

For a list of supported LDAP servers, refer to the Supported hardware and software Web site.

It is expected that other LDAP servers follow the LDAP specification. Support is limited to these specific directory servers only. You can use any other directory server by using the custom directory type in the list and by filling in the filters that are required for that directory.

To improve performance for LDAP searches, the default filters for IBM Tivoli Directory Server, Sun ONE, and Active Directory are defined such that when you search for a user, the result contains all the relevant information about the user (user ID, groups, and so on). As a result, the product does not call the LDAP server multiple times. This definition is possible only in these directory types, which support searches where the complete user information is obtained.

If you use the IBM Directory Server, select the **Ignore case for authorization** option. This option is required because when the group information is obtained from the user object attributes, the case is not the same as when you get the group information directly. For the authorization to work in this case, perform a case insensitive check and verify the requirement for the **Ignore case for authorization** option.

- **Using IBM Tivoli Directory Server as the LDAP server**

  To use IBM Tivoli Directory Server, formerly IBM Directory Server, select **IBM Tivoli Directory Server** as the directory type.

The difference between these two types is group membership lookup. It is recommended that you choose the IBM Tivoli Directory Server for optimum performance during runtime. In the IBM Tivoli Directory Server, the group membership is an operational attribute. With this attribute, a group membership lookup is done by enumerating the ibm-allGroups attribute for the entry. All group memberships, including the static groups, dynamic groups, and nested groups, can be returned with the ibm-allGroups attribute.

WebSphere Application Server supports dynamic groups, nested groups, and static groups in IBM Tivoli Directory Server using the ibm-allGroups attribute. To utilize this attribute in a security authorization application, use a case-insensitive match so that attribute values returned by the ibm-allGroups attribute are all in uppercase.

Important: It is recommended that you do not install IBM Tivoli Directory Server Version 6.0 on the same machine that you install WebSphere Application Server Version 6.1. IBM Tivoli Directory Server Version 6.0 includes WebSphere Application Server Express Version 5.1.1, which the directory server uses for its administrative console. Install the Web Administration tool Version 6.0 and WebSphere Application Server Express Version 5.1.1, which are both bundled with IBM Tivoli Directory Server Version 6.0, on a different machine from WebSphere Application Server Version 6.1. You cannot use WebSphere Application Server Version 6.1 as the administrative console for IBM Tivoli Directory Server. If IBM Tivoli Directory Server Version 6.0 and WebSphere Application Server Version 6.1 are installed on the same machine, you might encounter port conflicts.

If you must install IBM Tivoli Directory Server Version 6.0 and WebSphere Application Server Version 6.1 on the same machine, consider the following information:

- During the IBM Tivoli Directory Server installation process, you must select both the **Web Administration tool** and **WebSphere Application Server Express Version 5.1.1**.
- Install WebSphere Application Server Version 6.1.
- When you install WebSphere Application Server Version 6.1, change the port number for the application server.
- You might need to adjust the WebSphere Application Server environment variables on WebSphere Application Server Version 6.1 for *WAS_HOME* and *WAS_INSTALL_ROOT* (or *APP_SERVER_ROOT* for i5/OS). To change the variables using the administrative console, click **Environment > WebSphere Variables**.

- **Using a Lotus Domino Enterprise Server as the LDAP server**

  If you select the Lotus Domino Enterprise Server Version 6.5.4 or Version 7.0 and the attribute short name is not defined in the schema, you can take either of the following actions:
  - Change the schema to add the short name attribute.
  - Change the user ID map filter to replace the short name with any other defined attribute (preferably to UID). For example, change person:shortname to person:uid.

  The userID map filter is changed to use the uid attribute instead of the shortname attribute as the current version of Lotus Domino does not create the shortname attribute by default. If you want to use the shortname attribute, define the attribute in the schema and change the userID map filter.

  User ID Map : person:shortname

- **Using Sun ONE Directory Server as the LDAP server**

  You can select **Sun ONE Directory Server** for your Sun ONE Directory Server system. In Sun ONE Directory Server, the object class is the default groupOfUniqueName when you create a group. For better performance, WebSphere Application Server uses the User object to locate the user group membership from the nsRole attribute. Create the group from the role. If you want to use the groupOfUniqueName attribute to search groups, specify your own filter setting. Roles unify entries. Roles are designed to be more efficient and easier to use for applications. For example, an application

can locate the role of an entry by enumerating all the roles that are possessed by a given entry, rather than selecting a group and browsing through the members list. When using roles, you can create a group using a:

– Managed role
– Filtered role
– Nested role

All of these roles are computable by the nsRole attribute.

- **Using Microsoft Active Directory server as the LDAP server**

  To use Microsoft Active Directory as the LDAP server for authentication with WebSphere Application Server you must take specific steps. By default, Microsoft Active Directory does not permit anonymous LDAP queries. To create LDAP queries or to browse the directory, an LDAP client must bind to the LDAP server using the distinguished name (DN) of an account that belongs to the administrator group of the Windows system. A group membership search in the Active Directory is done by enumerating the memberof attribute for a given user entry, rather than browsing through the member list in each group. If you change the default behavior to browse each group, you can change the **Group Member ID Map** field from memberof:member to group:member.

The following steps describe how to set up Microsoft Active Directory as your LDAP server.

1. Determine the full distinguished name (DN) and password of an account in the administrators group.

   For example, if the Active Directory administrator creates an account in the Users folder of the Active Directory Users and Computers Windows control panel and the DNS domain is ibm.com, the resulting DN has the following structure:

   ```
   cn=<adminUsername>, cn=users, dc=ibm,
   dc=com
   ```

2. Determine the short name and password of any account in the Microsoft Active Directory.
3. Use the WebSphere Application Server administrative console to set up the information that is needed to use Microsoft Active Directory.
   a. Click **Security > Secure administration, applications, and infrastructure**.
   b. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
   c. Set up LDAP with Active Directory as the type of LDAP server. Based on the information that is determined in the previous steps, you can specify the following values on the LDAP settings panel:

      Primary administrative user name
      Specify the name of a user with administrative privileges that is defined in the registry. This user name is used to access the administrative console or used by wsadmin.

      Type Specify Active Directory

      Host Specify the domain name service (DNS) name of the machine that is running Microsoft Active Directory.

      Base distinguished name (DN)
      Specify the domain components of the DN of the account that is chosen in the first step. For example: dc=ibm, dc=com

      Bind distinguished name (DN)
      Specify the full distinguished name of the account that is chosen in the first step. For example: cn=*adminUsername*, cn=users, dc=ibm, dc=com

      Bind password
      Specify the password of the account that is chosen in the first step.

   d. Click **OK** and **Save** to save the changes to the master configuration.
4. Click **Security > Secure administration, applications, and infrastructure**.
5. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.

6. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

   Server user ID or administrative user on a Version 6.0.x node
   Specify the short name of the account that is chosen in the second step.

   Server user password
   Specify the password of the account that is chosen in the second step.
7. **Optional:** Set ObjectCategory as the filter in the Group member ID map field to improve LDAP performance.
   a. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings** .
   b. Add ;objectCategory:group to the end of the Group member ID map field.
8. Click **OK** and **Save** to save the changes to the master configuration.
9. Stop and restart the administrative server so that the changes take effect.

Locating a user's group memberships in Lightweight Directory Access Protocol:

WebSphere Application Server security can be configured to search group memberships directly or indirectly. It can also be configured to search only a static group, or it can be configured to search static groups, recursive or nested groups, and dynamic groups for some Lightweight Directory Access Protocol (LDAP) servers.

- Evaluate group memberships from user object directly.
  - Several popular LDAP servers enable user objects to contain information about the groups to which they belong such as Microsoft Active Directory Server, or eDirectory. Some user group memberships can be computable attributes from the user object such as IBM Directory Server or Sun ONE directory server. In some LDAP servers, this attribute can be used to include a user's dynamic group memberships, nesting group memberships, and static group memberships to locate all the group memberships from a single attribute.
  - For example, in IBM Directory Server all group memberships including the static groups, dynamic groups, and nested groups can be returned using the ibm-allGroups attribute. In Sun ONE, all roles, including managed roles, filtered roles, and nested roles, are calculated using the nsRole attribute. If an LDAP server has such an attribute in a User object to include dynamic groups, nested groups, and static groups, WebSphere Application Server security can be configured to use this attribute to support these groups.
- Evaluate group memberships from a Group object indirectly.
  - Some LDAP servers enable only Group objects, such as the Lotus Domino LDAP server to contain information about users. The LDAP server does not enable the User object to contain information about groups. For this type of LDAP server, group membership searches are performed by locating the user on the member list of groups. The member list evaluation is not currently used in the static group membership search for WebSphere Application Server.
- Use the direct method for searching group memberships if your LDAP server has an attribute in the User object to include group information. To use the direct method or the indirect method, enter the appropriate value in the Group Member ID map field on the Advanced LDAP Settings panel using the following methods.
  - objectclass:attribute pairs for the indirect method
  - attribute:attribute pairs for the direct method
- Use the sample entries of attribute:attribute pairs in Group member ID map fields. Note that the groupMembership attribute lists all the static groups for which a user is a member. This attribute is NOT updated whenever an object matches or does not match a dynamic group's filter. Please refer to the Novell eDirectory documentation for more information about the groupMembership attribute.
  - ibm-allGroups:member for IBM Directory server

  - nsRole:nsRole for Sun ONE directory, if groups are created with role inside Sun ONE
  - memberOf:member in Microsoft Active Directory Server
  - groupMembership:member for eDirectory
- Use the sample entries of objectClass:attribute pairs in the Group member ID map field.
  - dominoGroup:member for Lotus Domino
  - groupOfNames:member for eDirectory

While using the direct method, dynamic groups, recursive groups, and static groups can be returned as multiple values of a single attribute. For example, in IBM Directory Server all group memberships, including the static groups, dynamic groups, and nested groups, can be returned using the ibm-allGroups attribute. In Sun ONE, all roles, including managed roles, filtered roles, and nested roles, are calculated using the nsRole attribute. If an LDAP server can use the nsRole attribute, dynamic groups, nested groups, and static groups are all supported by WebSphere Application Server.

Some LDAP servers do not have recursive computing functionality. For example, although Microsoft Active Directory server has direct group search capability using the memberOf attribute, this attribute lists the groups beneath, which the group is directly nested only and does not contain the recursive list of nested predecessors. The Lotus Domino LDAP server only supports the indirect method to locate the group memberships for a user. You cannot obtain recursive group memberships from a Domino server directly. For LDAP servers without recursive searching capability, WebSphere Application Server security provides a recursive function that is enabled by clicking **Perform a Nested Group Search** in the Advanced LDAP user registry settings. Select this option only if your LDAP server does not provide recursive searches and you want a recursive search.

Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server:

Configure dynamic and nested groups to simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

To use dynamic and nested groups with WebSphere Application Server security, you must be running WebSphere Application Server Version 5.1.1 or later. Refer to “Dynamic and nested group support for the SunONE or iPlanet Directory Server” on page 1184 for more information on this topic.

1. In the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.
3. Select `SunONE` for the type of LDAP server.
4. Select the **Ignore case for authorization** option.
5. Under Additional Properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
6. Change the Group filter setting to `&(cn=%v)(objectclass=ldapsubentry))`.
7. Change the Group member ID map setting to `nsRole:nsRole`.
8. Click **Apply** or **OK** to validate the changes.

Configuring dynamic and nested group support for the IBM Tivoli Directory Server:

Configure dynamic and nested groups to simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

When creating groups, ensure that nested and dynamic group memberships work correctly.

1. In the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.

2. Under User account repository, click **Standalone LDAP registry**, and click **Configure**.
3. Select `IBM Tivoli Directory Server` for the type of LDAP server.
4. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
5. Change the Group filter value to `(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))`.
6. Change the Group member ID map value to `ibm-allGroups:member;ibm-allGroups:uniqueMember`.
7. Click **Apply** or **OK** to validate the changes.
8. Verify that Auxiliary object class field on the Add an LDAP entry panel for your IBM Tivoli Directory server has the appropriate value. When you create a nested group, the Auxiliary object class value is `ibm-nestedGroup`. When you create a dynamic group, the Auxiliary object class value is `ibm-dynamicGroup`.

Configuring standalone custom registries

Before you begin this task, implement and build the UserRegistry interface. For more information on developing standalone custom registries refer to Developing standalone custom registries. The following steps are required to configure standalone custom registries through the administrative console.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repositories, select **Standalone custom registry** and click **Configure**.
3. Enter a valid user name in the Primary administrative user name field. This ID is the security server ID, which is only used for WebSphere Application Server security and is not associated with the system process that runs the server. The server calls the local operating system registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry.
4. Enter the complete location of the dot-separated class name that implements the com.ibm.websphere.security.UserRegistry interface in the Custom registry class name field. For the sample, this file name is com.ibm.websphere.security.FileRegistrySample.

   The file exists in the WebSphere Application Server class path preferably in the *app_server_root*/lib/ext directory.

   Attention: The sample provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.
5. Add your custom registry class name to the class path. It is recommended that you add the Java Archive (JAR) file that contains your custom user registry implementation to the *app_server_root*/classes directory.
6. **Optional:** Select the **Ignore case for authorization** option for the authorization to perform a case insensitive check. Enabling this option is necessary only when your user registry is case insensitive and does not provide a consistent case when queried for users and groups.
7. Click **Apply** if you have any other additional properties to enter for the registry initialization.
8. **Optional:** Enter additional properties to initialize your implementation.
   a. Click **Custom properties > New**.
   b. Enter the property name and value.

      For the sample, enter the following two properties. It is assumed that the users.props file and the groups.props file are in the *customer_sample* directory under the product installation directory. You can place these properties in any directory that you choose and reference their locations through custom properties. However, make sure that the directory has the appropriate access permissions.

Property name	Property value
usersFile	${*USER_INSTALL_ROOT*}/*customer_sample* /users.props
groupsFile	${*USER_INSTALL_ROOT*}/*customer_sample* /groups.props

Samples of these two properties are available in “users.props file” on page 1128 and “groups.props file” on page 1128.

The **Description**, **Required**, and **Validation Expression** fields are not used and can remain blank.

WebSphere Application Server version 4-based custom user registry is migrated to the custom user registry based on the com.ibm.websphere.security.UserRegistry interface.

c. Click **Apply**.

d. Repeat this step to add other additional properties.

9. Click **Security > Secure administration, applications, and infrastructure**.
10. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone custom registry**, and click **Configure**.
11. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

    Server user ID or administrative user on a Version 6.0.x node
    Specify the short name of the account that is chosen in the second step.

    Server user password
    Specify the password of the account that is chosen in the second step.
12. Click **OK** and complete the required steps to turn on security.

This set of steps is required to set up the standalone custom registry and to enable security in WebSphere Application Server.

Note: The security component of WebSphere Application Server expands a selected list of variables when enabling security. See Variable settings for more detail.

1. Complete the remaining steps, if you are enabling security.
2. Validate the user and password. Save and synchronize in the cell environment.
3. After security is turned on, save, stop, and start all the product servers, including cell, nodes, and all of the application servers, for any changes to take effect. If the server comes up without any problems, the setup is correct.

Standalone custom registry settings:

Use this page to configure the standalone custom registry.

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

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone custom registry**, and click **Configure**.

After the properties are set in this panel, click **Apply**. Under Additional Properties, click **Custom properties** to include additional properties that the custom user registry requires.

Note: Custom properties might include information such as specifying lists of users or groups.

When security is enabled and any of these custom user registry settings change, go to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are

adding a Version 5.0.*x* or 6.0.*x* node to a Version 6.1 cell, you must ensure that the Version 5.*x* or Version 6.0.*x* server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Custom registry class name:

Specifies a dot-separated class name that implements the com.ibm.websphere.security.UserRegistry interface.

Put the custom registry class name in the class path. A suggested location is the *%install_root%*/lib/ext directory.

Data type: String
Default: com.ibm.websphere.security.FileRegistrySample

Ignore case for authorization:

Indicates that a case-insensitive authorization check is performed when you use the default authorization.

Default: Disabled
Range: Enabled or Disabled

Standalone custom registry wizard settings:

Use this page to provide the basic settings to connect the application server to an existing standalone custom registry.

To view this security wizard page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Standalone custom registry** option and click **Next**.

You can modify your standalone custom registry configuration by completing the following steps:

1. Click **Security > Security administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone custom registry**, and click **Configure**.
3. Enter additional properties to initialize your implementation
   - Click **Custom properties > New**.
   - Enter the property name and value. For the sample, enter the following two properties. It is assumed that the users.props file and the groups.props file are in the *customer_sample* directory under the product installation directory. You can place these properties in any directory that you choose and reference their locations through Custom properties. However, make sure that the directory has the appropriate access permissions.

Property name	Property value
usersFile	${*USER_INSTALL_ROOT*}/*customer_sample* /users.props
groupsFile	${*USER_INSTALL_ROOT*}/*customer_sample* /groups.props

     Samples of these two properties are available in “users.props file” on page 1128 and “groups.props file” on page 1128.

     The Description, Required, and Validation Expression fields are not used and can remain blank.

     WebSphere Application Server Version 4 based custom user registry is migrated to the custom user registry based on the com.ibm.websphere.security.UserRegistry interface.
   - Click **Apply**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

Custom registry class name:

Specifies a dot-separated class name that implements the com.ibm.websphere.security.UserRegistry interface.

Put the custom registry class name in the class path. A suggested location is the *%install_root%*/lib/ext directory.

Data type: String
Default: com.ibm.websphere.security.FileRegistrySample

Ignore case for authorization:

Indicates that a case-insensitive authorization check is performed when you use the default authorization.

Default:	Disabled
Range:	Enabled or Disabled

FileRegistrySample.java file:

This provides an example of the FileRegistrySample.java file.

The user and group information required by this sample is contained in the "users.props file" on page 1128 and "groups.props file" on page 1128 files.

Attention: The samples that are provided are intended to familiarize you with this feature. Do not use these samples in an actual production environment.

The contents of the `FileRegistrySample.java` file:

```
//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
// All Rights Reserved * Licensed Materials - Property of IBM
////--------------------------------------------------------------------
// This program may be used, run, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing.
//--------------------------------------------------------------------
//

// This sample is for the custom user registry feature in WebSphere
// Application Server.

import java.util.*;
import java.io.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.*;
/**
 *  The main purpose of this sample is to demonstrate the use of the
 *  custom user registry feature available in WebSphere Application Server. This
 *  sample is a file-based registry sample where the users and the groups
 *  information is listed in files (users.props and groups.props). As such
 *  simplicity and not the performance was a major factor. This
 *  sample should be used only to get familiarized with this feature. An
 *  actual implementation of a realistic registry should consider various
 *  factors like performance, scalability, thread safety, and so on.
 **/
public class FileRegistrySample implements UserRegistry {

   private static String USERFILENAME = null;
   private static String GROUPFILENAME = null;

   /** Default Constructor **/
   public FileRegistrySample() throws java.rmi.RemoteException {
   }

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param     props - The registry-specific properties with which to
   *                    initialize the custom registry
   * @exception CustomRegistryException
```

```
 *                   if there is any registry-specific problem
 **/
public void initialize(java.util.Properties props)
       throws CustomRegistryException {
    try {
       /* try getting the USERFILENAME and the GROUPFILENAME from
        * properties that are passed in (For example, from the
        * administrative console). Set these values in the administrative
        * console. Go to the special custom settings in the custom
        * user registry section of the Authentication panel.
        * For example:
        * usersFile   c:/temp/users.props
        * groupsFile  c:/temp/groups.props
        */
        if (props != null) {
           USERFILENAME = props.getProperty("usersFile");
           GROUPFILENAME = props.getProperty("groupsFile");
        }

    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    }

    if (USERFILENAME == null || GROUPFILENAME == null) {
       throw new CustomRegistryException("users/groups information missing");
    }

 }  /**
 * Checks the password of the user. This method is called to authenticate
 * a user when the user's name and password are given.
 *
 * @param  userSecurityName the name of user
 * @param  password the password of the user
 * @return a valid userSecurityName. Normally this is
 *         the name of same user whose password was checked
 *         but if the implementation wants to return any other
 *         valid userSecurityName in the registry it can do so
 * @exception CheckPasswordFailedException if userSecurityName/
 *            password combination does not exist
 *            in the registry
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public String checkPassword(String userSecurityName, String passwd)
    throws PasswordCheckFailedException,
           CustomRegistryException {
    String s,userName = null;
    BufferedReader in = null;

    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":",index+1);
             // check if the userSecurityName:passwd combination exists
             if ((s.substring(0,index)).equals(userSecurityName) &&
                     s.substring(index+1,index1).equals(passwd)) {
                // Authentication successful, return the userID.
                userName = userSecurityName;
                break;
             }
```

```
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }


   if (userName == null) {
      throw new PasswordCheckFailedException("Password check failed for user: "
      + userSecurityName);
   }

   return userName;
}  /**
* Maps an X.509 format certificate to a valid user in the registry.
* This is used to map the name in the certificate supplied by a browser
* to a valid userSecurityName in the registry
*
* @param     cert the X509 certificate chain
* @return    The mapped name of the user userSecurityName
* @exception CertificateMapNotSupportedException if the
*            particular certificate is not supported.
* @exception CertificateMapFailedException if the mapping of
*            the certificate fails.
* @exception CustomRegistryException if there is any registry
*            -specific problem
**/
public String mapCertificate(X509Certificate[] cert)
   throws CertificateMapNotSupportedException,
          CertificateMapFailedException,
          CustomRegistryException {
   String name=null;
   X509Certificate cert1 = cert[0];
   try {
      // map the SubjectDN in the certificate to a userID.
      name = cert1.getSubjectDN().getName();
   } catch(Exception ex) {
      throw new CertificateMapNotSupportedException(ex.getMessage(),ex);
   }

   if(!isValidUser(name)) {
      throw new CertificateMapFailedException("user: " + name
      + " is not valid");
   }
   return name;
}  /**
* Returns the realm of the registry.
*
* @return the realm. The realm is a registry-specific string
* indicating the realm or domain for which this registry
* applies. For example, for OS/400 or AIX this would be
* the host name of the system whose user registry this
* object represents. If null is returned by this method,
* realm defaults to the value of "customRealm". It is
* recommended that you use your own value for realm.
*
* @exception CustomRegistryException if there is any registry-
* specific problem
**/
public String getRealm()
   throws CustomRegistryException {
```

```
   String name = "customRealm";
   return name;
}  /**
* Gets a list of users that match a pattern in the registry.
* The maximum number of users returned is defined by the limit
* argument.
* This method is called by the administrative console and scripting
* (command line) to make the users in the registry available for
* adding them (users) to roles.
*
* @param      pattern the pattern to match. (For example, a* will
*             match  all userSecurityNames starting with a)
* @param      limit the maximum number of users that should be
*             returned. This is very useful in situations where
*             there are thousands of users in the registry and
*             getting all of them at once is not practical. The
*             default is 100. A value of 0 implies get all the
*             users and hence must be used with care.
* @return     a Result object that contains the list of users
*             requested and a flag to indicate if  more users
*             exist.
* @exception  CustomRegistryException if there is any registry-
*             specific problem
**/
public Result getUsers(String pattern, int limit)
   throws CustomRegistryException {
   String s;
   BufferedReader in = null;
   List allUsers = new ArrayList();
   Result result = new Result();
   int count = 0;
   int newLimit = limit+1;
   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            String user = s.substring(0,index);
            if (match(user,pattern)) {
               allUsers.add(user);
               if (limit !=0 && ++count == newLimit) {
                  allUsers.remove(user);
                  result.setHasMore();
                  break;
               }
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   result.setList(allUsers);
   return result;
}  /**
* Returns the display name for the user specified by
*  userSecurityName.
*
* This method may be called only when the user information
* is displayed (information purposes only, for example, in
```

```
* the administrative console) and hence not used in the actual
* authentication or authorization purposes. If there are no
* display names in the registry return null or empty string.
*
* In WebSphere Application Server 4.x custom registry, if you
* had a display name for the user and if it was different from the
* security name, the display name was returned for the EJB
* methods getCallerPrincipal() and the servlet methods
* getUserPrincipal() and  getRemoteUser().
* In WebSphere Application Server Version 5.x and later, for the
* same methods, the security name will be returned by default.
* This is the recommended way as the display name is not unique
* and might create security holes. However, for backward
* compatibility if you need the display name to be returned
* set the property WAS_UseDisplayName to true.
*
*See the Information Center documentation for more information.
*
* @param     userSecurityName the name of the user.
* @return    the display name for the user. The display
*            name is a registry-specific string that
*            represents a descriptive, not necessarily
*            unique, name for a user. If a display name
*            does not exist return null or empty string.
* @exception EntryNotFoundException if userSecurityName
*            does not exist.
* @exception CustomRegistryException if there is any registry-
 *           specific problem
**/
public String getUserDisplayName(String userSecurityName)
   throws CustomRegistryException,
          EntryNotFoundException {

   String s,displayName = null;
   BufferedReader in = null;

   if(!isValidUser(userSecurityName)) {
      EntryNotFoundException nsee = new EntryNotFoundException("user: "
      + userSecurityName + " is not valid");
      throw nsee;
   }

   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.lastIndexOf(":");
            if ((s.substring(0,index)).equals(userSecurityName)) {
               displayName = s.substring(index1+1);
               break;
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(), ex);
   } finally {
      fileClose(in);
   }

   return displayName;
}
```

```
/**
 * Returns the unique ID for a userSecurityName. This method is called
 * when creating a credential for a user.
 *
 * @param    userSecurityName - The name of the user.
 * @return   The unique ID of the user. The unique ID for a user
 *           is the stringified form of some unique, registry-specific,
 *           data that serves to represent the user. For example, for
 *           the UNIX user registry, the unique ID for a user can be
 *           the UID.
 * @exception EntryNotFoundException if userSecurityName does not
 *            exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public String getUniqueUserId(String userSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {

    String s,uniqueUsrId = null;
    BufferedReader in = null;
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(0,index)).equals(userSecurityName)) {
                int index2 = s.indexOf(":", index1+1);
                uniqueUsrId = s.substring(index1+1,index2);
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    if (uniqueUsrId == null) {
       EntryNotFoundException nsee =
       new EntryNotFoundException("Cannot obtain uniqueId for user: "
       + userSecurityName);
       throw nsee;
    }

    return uniqueUsrId;
 }  /**
 * Returns the name for a user given its unique ID.
 *
 * @param      uniqueUserId  - The unique ID of the user.
 * @return     The userSecurityName of the user.
 * @exception  EntryNotFoundException if the unique user ID does not exist.
 * @exception  CustomRegistryException if there is any registry-specific
 *             problem
 **/
 public String getUserSecurityName(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,usrSecName = null;
```

```
   BufferedReader in = null;
   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            int index2 = s.indexOf(":", index1+1);
            if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
               usrSecName = s.substring(0,index);
               break;
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(), ex);
   } finally {
      fileClose(in);
   }

   if (usrSecName == null) {
      EntryNotFoundException ex =
         new EntryNotFoundException("Cannot obtain the
         user securityName for " + uniqueUserId);
      throw ex;
   }

   return usrSecName;

}  /**
* Determines if the userSecurityName exists in the registry
*
* @param     userSecurityName - The name of the user
* @return    True if the user is valid; otherwise false
* @exception CustomRegistryException if there is any registry-
*            specific problem
* @exception RemoteException as this extends java.rmi.Remote
*            interface
**/
public boolean isValidUser(String userSecurityName)
   throws CustomRegistryException {
   String s;
   boolean isValid = false;
   BufferedReader in = null;
   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            if ((s.substring(0,index)).equals(userSecurityName)) {
               isValid=true;
               break;
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(), ex);
   } finally {
      fileClose(in);
   }
```

```
    return isValid;
 }
/**
 * Gets a list of groups that match a pattern in the registry
 * The maximum number of groups returned is defined by the
 * limit argument. This method is called by administrative console
 * and scripting (command line) to make available the groups in
 * the registry for adding them (groups) to roles.
 *
 * @param       pattern the pattern to match. (For example, a* matches
 *              all groupSecurityNames starting with a)
 * @param       Limits the maximum number of groups to return
 *              This is very useful in situations where there
 *              are thousands of groups in the registry and getting all
 *              of them at once is not practical. The default is 100.
 *              A value of 0 implies get all the groups and hence must
 *              be used with care.
 * @return      A Result object that contains the list of groups
 *              requested and a flag to indicate if more groups exist.
 * @exception CustomRegistryException if there is any registry-specific
 *              problem
 **/
 public Result getGroups(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allGroups = new ArrayList();      Result result = new Result();
    int count = 0;
    int newLimit = limit+1;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             String group = s.substring(0,index);
             if (match(group,pattern)) {
                allGroups.add(group);
                if (limit !=0 && ++count == newLimit) {
                   allGroups.remove(group);
                   result.setHasMore();
                   break;
                }
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    result.setList(allGroups);
    return result;
 }

 /**
  * Returns the display name for the group specified by groupSecurityName.
  * For this version of WebSphere Application Server, the only usage of
  * this method is by the clients (administrative console and scripting)
  * to present a descriptive name of the user if it exists.
  *
  * @param groupSecurityName the name of the group.
```

```
 * @return  the display name for the group. The display name
 *          is a registry-specific string that represents a
 *          descriptive, not necessarily unique, name for a group.
 *          If a display name does not exist return null or empty
 *          string.
 * @exception EntryNotFoundException if groupSecurityName does
 *          not exist.
 * @exception CustomRegistryException if there is any registry-
 *          specific problem
 **/
 public String getGroupDisplayName(String groupSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,displayName = null;
    BufferedReader in = null;

    if(!isValidGroup(groupSecurityName)) {
       EntryNotFoundException nsee = new EntryNotFoundException("group: "
       + groupSecurityName + " is not valid");
       throw nsee;
    }

    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.lastIndexOf(":");
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                displayName = s.substring(index1+1);
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return displayName;
 }

/**
 * Returns the Unique ID for a group.

 * @param     groupSecurityName the name of the group.
 * @return    The unique ID of the group. The unique ID for
 *            a group is the stringified form of some unique,
 *            registry-specific, data that serves to represent
 *            the group. For example, for the UNIX user registry,
 *            the unique ID might be the GID.
 * @exception EntryNotFoundException if groupSecurityName does
 *            not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUniqueGroupId(String groupSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
```

```
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                uniqueGrpId = s.substring(index+1,index1);
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    if (uniqueGrpId == null) {
       EntryNotFoundException nsee =
       new EntryNotFoundException("Cannot obtain the uniqueId for group: "
       + groupSecurityName);
       throw nsee;
    }

    return uniqueGrpId;
 }

/**
 * Returns the Unique IDs for all the groups that contain the unique ID
 * of a user. Called during creation of a user's credential.
 *
 * @param     uniqueUserId the unique ID of the user.
 * @return    A list of all the group unique IDs that the unique user
 *            ID belongs to. The unique ID for an entry is the
 *            stringified form of some unique, registry-specific, data
 *            that serves to represent the entry.  For example, for the
 *            UNIX user registry, the unique ID for a group might be
 *            the GID and the Unique ID for the user might be the UID.
 * @exception EntryNotFoundException if uniqueUserId does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *            problem
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    List uniqueGrpIds=new ArrayList();
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             int index2 = s.indexOf(":", index1+1);
             if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                int lastIndex = s.lastIndexOf(":");
                String subs = s.substring(index2+1,lastIndex);
                StringTokenizer st1 = new StringTokenizer(subs, ",");
```

```
                while (st1.hasMoreTokens())
                   uniqueGrpIds.add(st1.nextToken());
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return uniqueGrpIds;
 }

/**
 * Returns the name for a group given its unique ID.
 *
 * @param     uniqueGroupId the unique ID of the group.
 * @return    The name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does
 *            not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,grpSecName = null;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
                grpSecName = s.substring(0,index);
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    if (grpSecName == null) {
       EntryNotFoundException ex =
          new EntryNotFoundException("Cannot obtain the group
         security name for: " + uniqueGroupId);
       throw ex;
    }

    return grpSecName;

 }

/**
 * Determines if the groupSecurityName exists in the registry
 *
```

```
 * @param     groupSecurityName the name of the group
 * @return    True if the groups exists; otherwise false
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException {
    String s;
    boolean isValid = false;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                isValid=true;
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return isValid;
 }

/**
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by the administrative console and scripting
 * (command line) to verify that the user entered for RunAsRole mapping
 * belongs to that role in the roles to user mapping. Initially, the
 * check is done to see  if the role contains the user. If the role does
 * not contain the user explicitly, this method is called to get the groups
 * that this user belongs to so that a check can be made on the groups that
 * the role contains.
 *
 * @param     userSecurityName the name of the user
 * @return    A list of all the group securityNames that the user
 *            belongs to.
 * @exception EntryNotFoundException if user does not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 * @exception RemoteException as this extends the java.rmi.Remote
 *            interface
 **/
 public List getGroupsForUser(String userName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s;
    List grpsForUser = new ArrayList();
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             StringTokenizer st = new StringTokenizer(s, ":");
```

```
            for (int i=0; i<2; i++)
               st.nextToken();
            String subs = st.nextToken();
            StringTokenizer st1 = new StringTokenizer(subs, ",");
            while (st1.hasMoreTokens()) {
               if((st1.nextToken()).equals(userName)) {
                  int index = s.indexOf(":");
                  grpsForUser.add(s.substring(0,index));
               }
            }
         }
      }
   } catch (Exception ex) {
      if (!isValidUser(userName)) {
         throw new EntryNotFoundException("user: " + userName
         + " is not valid");
      }
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   return grpsForUser;
}

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the
 * limit argument.
 *
 * This method is being used by the WebSphere Application
 * Server Enterprise process choreographer (Enterprise) when
 * staff assignments are modeled using groups.
 *
 * In rare situations, if you are working with a registry where
 * getting all the users from any of your groups is not practical
 * (for example if there are a large number of users) you can create
 * the NotImplementedException for that particular group. Make sure
 * that if the process choreographer is installed (or if installed later)
 * the staff assignments are not modeled using these particular groups.
 * If there is no concern about returning the users from groups
 * in the registry it is recommended that this method be implemented
 * without creating the NotImplemented exception.
 * @param         groupSecurityName the name of the group
 * @param         Limits the maximum number of users that should be
 *                returned. This is very useful in situations where there
 *                are lots of users in the registry and getting all of
 *                them at once is not practical. A value of 0 implies
 *                get all the users and hence must be used with care.
 * @return        A Result object that contains the list of users
 *                requested and a flag to indicate if more users exist.
 * @deprecated    This method will be deprecated in future.
 * @exception     NotImplementedException create this exception in rare
 *                situations if it is not practical to get this information
 *                for any of the group or groups from the registry.
 * @exception     EntryNotFoundException if the group does not exist in
 *                the registry
 * @exception     CustomRegistryException if there is any registry-specific
 *                problem
 **/
public Result getUsersForGroup(String groupSecurityName, int limit)
   throws NotImplementedException,
```

```
            EntryNotFoundException,
            CustomRegistryException {
        String s, user;
        BufferedReader in = null;
        List usrsForGroup = new ArrayList();
        int count = 0;
        int newLimit = limit+1;
        Result result = new Result();

        try {
           in = fileOpen(GROUPFILENAME);
           while ((s=in.readLine())!=null)
           {
              if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                 int index = s.indexOf(":");
                 if ((s.substring(0,index)).equals(groupSecurityName))
                 {
                    StringTokenizer st = new StringTokenizer(s, ":");
                    for (int i=0; i<2; i++)
                       st.nextToken();
                    String subs = st.nextToken();
                    StringTokenizer st1 = new StringTokenizer(subs, ",");
                    while (st1.hasMoreTokens()) {
                       user = st1.nextToken();
                       usrsForGroup.add(user);
                       if (limit !=0 && ++count == newLimit) {
                          usrsForGroup.remove(user);
                          result.setHasMore();
                          break;
                       }
                    }
                 }
              }
           }
        } catch (Exception ex) {
           if (!isValidGroup(groupSecurityName)) {
              throw new EntryNotFoundException("group: "
              + groupSecurityName
              + " is not valid");
           }
           throw new CustomRegistryException(ex.getMessage(),ex);
        } finally {
           fileClose(in);
        }

        result.setList(usrsForGroup);
        return result;
     }

    /**
     * This method is implemented internally by the WebSphere Application
     * Server code in this release. This method is not called for the custom
     * registry implementations for this release. Return null in the
     * implementation.
     *
     **/
     public com.ibm.websphere.security.cred.WSCredential
           createCredential(String userSecurityName)
           throws CustomRegistryException,
                  NotImplementedException,
                  EntryNotFoundException {

        // This method is not called.
```

```
      return null;
   }

   // private methods
   private BufferedReader fileOpen(String fileName)
      throws FileNotFoundException {
      try {
         return new BufferedReader(new FileReader(fileName));
      } catch(FileNotFoundException e) {
         throw e;
      }
   }

   private void fileClose(BufferedReader in) {
      try {
         if (in != null) in.close();
      } catch(Exception e) {
         System.out.println("Error closing file" + e);
      }
   }

   private boolean match(String name, String pattern) {
      RegExpSample regexp = new RegExpSample(pattern);
      boolean matches = false;
      if(regexp.match(name))
          matches = true;
      return matches;
   }
}


//-------------------------------------------------------------------------
// The program provides the Regular Expression implementation
// used in the sample for the custom user registry (FileRegistrySample).
// The pattern matching in the sample uses this program to search for the
// pattern (for users and groups).
//-------------------------------------------------------------------------

class RegExpSample
{

    private boolean match(String s, int i, int j, int k)
    {
        for(; k < expr.length; k++)
label0:
            {
                Object obj = expr[k];
                if(obj == STAR)
                {
                    if(++k >= expr.length)
                        return true;
                    if(expr[k] instanceof String)
                    {
                        String s1 = (String)expr[k++];
                        int l = s1.length();
                        for(; (i = s.indexOf(s1, i)) >= 0; i++)
                            if(match(s, i + l, j, k))
                                return true;

                        return false;
                    }
                    for(; i < j; i++)
                        if(match(s, i, j, k))
```

```
                        return true;

                return false;
            }
            if(obj == ANY)
            {
                if(++i > j)
                    return false;
                break label0;
            }
            if(obj instanceof char[][])
            {
                if(i >= j)
                    return false;
                char c = s.charAt(i++);
                char ac[][] = (char[][])obj;
                if(ac[0] == NOT)
                {
                    for(int j1 = 1; j1 < ac.length; j1++)
                        if(ac[j1][0] <= c && c <= ac[j1][1])
                            return false;

                    break label0;
                }
                for(int k1 = 0; k1 < ac.length; k1++)
                    if(ac[k1][0] <= c && c <= ac[k1][1])
                        break label0;

                return false;
            }
            if(obj instanceof String)
            {
                String s2 = (String)obj;
                int i1 = s2.length();
                if(!s.regionMatches(i, s2, 0, i1))
                    return false;
                i += i1;
            }
        }

    return i == j;
}

public boolean match(String s)
{
    return match(s, 0, s.length(), 0);
}

public boolean match(String s, int i, int j)
{
    return match(s, i, j, 0);
}

public RegExpSample(String s)
{
    Vector vector = new Vector();
    int i = s.length();
    StringBuffer stringbuffer = null;
    Object obj = null;
    for(int j = 0; j < i; j++)
    {
        char c = s.charAt(j);
        switch(c)
```

```
{
case 63: /* '?' */
    obj = ANY;
    break;

case 42: /* '*' */
    obj = STAR;
    break;

case 91: /* '[' */
    int k = ++j;
    Vector vector1 = new Vector();
    for(; j < i; j++)
    {
        c = s.charAt(j);
        if(j == k && c == '^')
        {
            vector1.addElement(NOT);
            continue;
        }
        if(c == '\\')
        {
            if(j + 1 < i)
                c = s.charAt(++j);
        }
        else
        if(c == ']')
            break;
        char c1 = c;
        if(j + 2 < i && s.charAt(j + 1) == '-')
            c1 = s.charAt(j += 2);
        char ac1[] = {
            c, c1
        };
        vector1.addElement(ac1);
    }

    char ac[][] = new char[vector1.size()][];
    vector1.copyInto(ac);
    obj = ac;
    break;

case 92: /* '\\' */
    if(j + 1 < i)
        c = s.charAt(++j);
    break;

}
if(obj != null)
{
    if(stringbuffer != null)
    {
        vector.addElement(stringbuffer.toString());
        stringbuffer = null;
    }
    vector.addElement(obj);
    obj = null;
}
else
{
    if(stringbuffer == null)
        stringbuffer = new StringBuffer();
    stringbuffer.append(c);
```

```
            }
        }

        if(stringbuffer != null)
            vector.addElement(stringbuffer.toString());
        expr = new Object[vector.size()];
        vector.copyInto(expr);
    }

    static final char NOT[] = new char[2];
    static final Integer ANY = new Integer(0);
    static final Integer STAR = new Integer(1);
    Object expr[];

}
```

users.props file:

This example presents the format for the `users.props` file.

Attention: The sample that is provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2005
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:passwd:uid:gids:display name
# where name   = userId/userName of the user
#       passwd = password of the user
#       uid    = uniqueId of the user
#       gid    = groupIds of the groups that the user belongs to
#       display name = a (optional) display name for the user.
bob:bob1:123:567:bob
dave:dave1:234:678:
jay:jay1:345:678,789:Jay-Jay
ted:ted1:456:678:Teddy G
jeff:jeff1:222:789:Jeff
vikas:vikas1:333:789:vikas
bobby:bobby1:444:789:
```

groups.props file:

The following example illustrates the format for the `groups.props` file.

Attention: The sample provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2005
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:gid:users:display name
# where name   = groupId of the group
#       gid    = uniqueId of the group
#       users  = list of all the userIds that the group contains
#       display name = a (optional) display name for the group.
admins:567:bob:Administrative group
operators:678:jay,ted,dave:Operators group
users:789:jay,jeff,vikas,bobby:
```

Managing the realm in a federated repository configuration

Follow this topic to manage the realm in a federated repository configuration.

The realm can consist of identities in:

- The file-based repository that is built into the system
- One or more external repositories
- Both the built-in, file-based repository and in one or more external repositories

Before you configure your realm, review "Limitations of federated repositories" on page 1131.

1. Configure your realm by using one of the following topics. You might be configuring your realm for the first time or changing an existing realm configuration.
   - "Using a single built-in, file-based repository in a new configuration under Federated repositories" on page 1132
   - "Changing a federated repository configuration to include a single built-in, file-based repository only" on page 1134
   - "Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories" on page 1135
   - "Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only" on page 1136
   - "Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 1137
   - "Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration" on page 1138
2. Configure supported entity types using the steps described in "Configuring supported entity types in a federated repository configuration" on page 1161. You must configure supported entity types before you can manage this account with Users and Groups. The Base entry for the default parent determines the repository location where entities of the specified type are placed on a create operation.
3. **Optional:** Use one or more of the following tasks to extend the capabilities of storing data and attributes in your realm:
   a. Configure an entry mapping repository using the steps described in "Configuring an entry mapping repository in a federated repository configuration" on page 1158. An entry mapping repository is used to store data for managing profiles on multiple repositories.
   b. Configure a property extension repository using the steps described in "Configuring a property extension repository in a federated repository configuration" on page 1147. A property extension repository is used to store attributes that cannot be stored in your Lightweight Directory Access Protocol (LDAP) server.
   a. Set up a database repository using wsadmin commands as described in "Setting up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands" on page 1151
4. **Optional:** Use one or more of the following advanced user tasks to extend the capabilities of LDAP repositories in your realm:
   - "Increasing the performance of the federated repository configuration" on page 1166
   - "Configuring Lightweight Directory Access Protocol entity types in a federated repository configuration" on page 1170
   - "Configuring group attribute definition settings in a federated repository configuration" on page 1172
5. **Optional:** Manage repositories that are configured in your system by following the steps described in "Managing repositories in a federated repository configuration" on page 1164.
6. **Optional:** Add an external repository into your realm by following the steps described in "Adding an external repository in a federated repository configuration" on page 1146.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Realm configuration settings:

Use this page to manage the realm. The realm can consist of identities in the file-based repository that is built into the system, in one or more external repositories, or in both the built-in, file-based repository and one or more external repositories.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

A single built-in, file-based repository is built into the system and included in the realm by default.

You can configure one or more Lightweight Directory Access Protocol (LDAP) repositories to store identities in the realm. Click **Add base entry to realm** to specify a repository configuration and a base entry into the realm. You can configure multiple different base entries into the same repository.

Click **Remove** to remove selected repositories from the realm. Repository configurations and contents are not destroyed. The following restrictions apply:

- The realm must always contain at least one base entry; therefore, you cannot remove every entry.
- If you plan to remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are adding a Version 5.0.*x* or 6.0.*x* node to a Version 6.1 cell, you must ensure that the Version 5.*x* or Version 6.0.*x* server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Ream name:

Specifies the name of the realm. You can change the realm name.

Primary administrative user name:

Specifies the name of the user with administrative privileges that is defined in the repository, for example, adminUser.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Ignore case for authorization:

Specifies that a case-insensitive authorization check is performed.

If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option.

Base entry:

Specifies the base entry within the realm. This entry and its descendents are part of the realm.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository type:

Specifies the repository type, such as File or LDAP.

Limitations of federated repositories:

This topic outlines known limitations and important information for configuring federated repositories.

Configuring federated repositories in a mixed-version environment

In a mixed-version deployment manager cell that contains both Version 6.1.x and Version 5.x or 6.0.x nodes, the following limitations apply for configuring federated repositories:

- You can configure only one Lightweight Directory Access Protocol (LDAP) repository under federated repositories, and the repository must be supported by Version 5.x or 6.0.x.
- You can specify a realm name that is compatible with prior versions only. The host name and the port number represent the realm for the LDAP server in a mixed-version nodes cell. For example, machine1.austin.ibm.com:389.
- You must configure a stand-alone LDAP registry; the LDAP information in both the stand-alone LDAP registry and the LDAP repository under the federated repositories configuration must match. During node synchronization, the LDAP information from the stand-alone LDAP registry propagates to the Version 5.x or 6.0.x nodes.

  Important: Before node synchronization, verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. Do not set the stand-alone LDAP registry as the current realm definition.
- You cannot configure an entry mapping repository or a property extension repository in a mixed-version deployment manager cell.

Coexisting with Tivoli Access Manager

For Tivoli Access Manager to coexist with a federated repositories configuration, the following limitations apply:

- You can configure only one LDAP repository under federated repositories, and that LDAP repository configuration must match the LDAP server configuration under Tivoli Access Manager.
- The distinguished name for the realm base entry must match the LDAP distinguished name (DN) of the base entry within the repository. In WebSphere Application Server, Tivoli Access Manager recognizes the LDAP user ID and LDAP DN for both authentication and authorization. The federated repositories configuration does not include additional mappings for the LDAP user ID and DN.
- The federated repositories functionality does not recognize the metadata that is specified by Tivoli Access Manager. When users and groups are created under user and group management, they are not formatted using the Tivoli Access Manager metadata. The users and groups must be manually imported into Tivoli Access Manager before you use them for authentication and authorization.

Using a single built-in, file-based repository in a new configuration under Federated repositories:

Follow this task to use a single built-in, file-based repository in a new configuration under Federated repositories.

To use the default configuration under Federated repositories that includes a single built-in, file-based repository only, you need to know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

Restriction: Client certificate login is not supported in a realm that includes a single built-in, file-based repository or a single built-in, file-based repository with other repositories.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Leave the Realm name field value as `defaultWIMFileBasedRealm`.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Leave the **Ignore case for authorization** option enabled.

6. Click **OK**.
7. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, the panel does not display. If a built-in, file-based repository is included, complete the following steps:
   a. Supply a password for the administrative user in the Password field.
   b. Confirm the password of the primary administrative user in the Confirm password field.
   c. Click **OK**.

After completing these steps, your new configuration under Federated repositories includes a single built-in, file-based repository only.
1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 1161.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps, as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Administrative user password settings:

Use this page to set a password for the administrative user who manages the product resources and user accounts.

To view this administrative console page, complete the following steps:
1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. If your federated repository configuration includes a built-in, file-based repository, then the **Administrative user password** panel displays when changes are applied.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Password:

Specifies the password of the administrative user who manages the product resources and user accounts.

Confirm password:

Confirms the password of the administrative user who manages the product resources and user accounts.

Federated repository wizard settings:

Use this security wizard page to complete the basic requirements to connect the application server to a federated repository.

To view this security wizard page, complete the following steps

1. Click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Federated repositories** option and click **Next**.

You can modify your federated repository configuration by completing the following steps:

1. Click **Security > Security administration, applications, and infrastructure**.
2. Under User account repository, select Federated repository and click **Configure**.

Note: This wizard is used for the initial configuration of a built-in, file-based repository. The user name and password do not have to be in the federated repository because they will be created. If you have previously configured federated repositories, do not use the Security configuration wizard to modify your configuration. Instead, modify your configuration using the Federated repositories selection under User account repository on the Secure administration, applications, and infrastructure panel.

Primary administrative user name:

Specifies the name of the user with administrative privileges that is defined in the repository, for example, adminUser.

Password:

Specifies the password of the administrative user who manages the product resources and user accounts.

Confirm password:

Confirms the password of the administrative user who manages the product resources and user accounts.

Changing a federated repository configuration to include a single built-in, file-based repository only:

Follow this task to change your federated repository configuration to include a single built-in, file-based repository only.

To change your federated repository configuration to include a single built-in, file-based repository only, you need to know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

Restriction: Client certificate login is not supported in a realm that includes a single built-in, file-based repository or a single built-in, file-based repository with other repositories.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. If the realm contains a single built-in, file-based repository only, you must specify `defaultWIMFileBasedRealm` as the realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Enable the **Ignore case for authorization** option.
6. Click **Use built-in repository** if the built-in, file-based repository is not listed in the collection.
7. Select all repositories in the collection that are not of type File and click **Remove**.

8. Click **OK**.
9. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, it does not display. If a built-in, file-based repository is included, complete the following steps:
   a. Supply a password for the primary administrative user in the Password field.
   b. Confirm the password of the primary administrative user in the Confirm password field.
   c. Click **OK**.

After completing these steps, your federated repository configuration, which includes a single built-in, file-based repository only, is configured.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in "Configuring supported entity types in a federated repository configuration" on page 1161.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps, as specified in "Enabling security for the realm" on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories:

Follow this task to configure a single, Lightweight Directory Access Protocol (LDAP) repository in a new configuration under Federated repositories.

To configure an LDAP repository in a new configuration under Federated repositories, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as the WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log in to the administrative console after you turn on security. You can use other users to log in, if those users are part of the administrative roles.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. On the Federated repositories panel, complete the following steps:
   a. Enter the name of the realm in the Realm name field. You can change the existing realm name.
   b. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
   c. **Optional:** Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.

d. Click **Add base entry to realm** to add a base entry that uniquely identifies the external repository in the realm. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 1146.

4. On the Federated repositories panel, complete the following steps:
   a. Select the built-in, file-based repository in the collection, and click **Remove**.

      Restriction: Before you remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.

   b. Click **OK**.

After completing these steps, your new configuration under Federated repositories includes a single, LDAP repository only.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 1161.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only:

Follow this task to change your federated repository configuration to include a single, Lightweight Directory Access Protocol repository (LDAP) repository only.

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log into the administrative console after you turn on security. You can use other users to log in if those users are part of the administrative roles.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.

4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. **Optional:** Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

   Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.
6. **Optional:** Click **Add base entry to realm** if the LDAP repository that you need is not contained in the collection. Then complete the steps in "Adding an external repository in a federated repository configuration" on page 1146.
7. On the Federated repositories panel, complete the following steps:
   a. **Optional:** Select the repositories in the collection that you do not need in the realm and click **Remove**.

      Restriction: The realm must always contain at least one base entry; therefore, you cannot remove every entry.
   b. Click **OK**.

After completing these steps, your federated repository configuration, which includes a single LDAP repository only, is configured.
1. Before you can manage this account with Users and Groups, configure supported entity types as described in "Configuring supported entity types in a federated repository configuration" on page 1161.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration:

Follow this task to configure multiple Lightweight Directory Access Protocol (LDAP) repositories in a federated repository configuration.

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log into the administrative console after you turn on security. You can use other users to log in if those users are part of the administrative roles.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. **Optional:** Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

   Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.
6. **Optional:** Click **Add base entry to realm** if the LDAP repository that you need is not listed in the collection. Then complete the steps in "Adding an external repository in a federated repository configuration" on page 1146.
7. On the Federated repositories panel, complete the following steps:
   a. **Optional:** Repeat step 6 if the LDAP repository that you need is not listed in the collection.
   b. **Optional:** Select the repositories in the collection that you do not need in the realm and click **Remove**. The following restrictions apply:
      - The realm must always contain at least one base entry; therefore, you cannot remove every entry.
      - If you plan to remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.
   c. Click **OK**.

After completing these steps, your federated repository configuration, which includes multiple LDAP repositories, is configured.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in "Configuring supported entity types in a federated repository configuration" on page 1161.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration:

Follow this task to configure a single built-in, file-based repository and multiple Lightweight Directory Access Protocol (LDAP) repositories in a federated repository configuration.

To configure a built-in, file-based repository in a federated repository configuration, you must know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log in to the administrative console after you turn on security. You can use other users to log in if those users are part of the administrative roles.

Restriction: Client certificate login is not supported in a realm that includes a single built-in, file-based repository or a single built-in, file-based repository with other repositories.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.

   Restriction: When you configure multiple repositories that includes a single built-in, file-based repository, the primary administrative user name must exist in the file-based repository. If the primary administrative user name does not exist in the file-based repository, then the name is created in the file-based repository. The primary administrative user name cannot exist in other repositories.
5. Select the **Ignore case for authorization** option.

   Attention: When the realm includes a built-in, file-based repository, you must enable the **Ignore case for authorization** option.
   When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

   Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.
6. **Optional:** Click **Add base entry to realm** if the LDAP repository that you need is not contained in the collection. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 1146.
7. On the Federated repositories panel, complete the following steps:
   a. **Optional:** Repeat step 6 if the LDAP repository that you need is not listed in the collection.
   b. Click **Use built-in repository** if the built-in, file-based repository is not listed in the collection.
   c. **Optional:** Select the repositories in the collection that you do not need in the realm and click **Remove**.

      Restriction: The realm must always contain at least one base entry; therefore, you cannot remove every entry.
   d. Click **OK**.

8. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, the panel does not display. If a built-in, file-based repository is included, complete the following steps:
   a. Supply a password for the administrative user in the Password field.
   b. Confirm the password of the primary administrative user in the Confirm password field.
   c. Click **OK**.

After completing these steps, your federated repository configuration, which includes a single built-in, file-based repository and one or more LDAP repositories, is configured.
1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 1161.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring Lightweight Directory Access Protocol in a federated repository configuration:

Follow this topic to configure Lightweight Directory Access Protocol (LDAP) settings in a federated repository configuration.

You have chosen among various ways to configure LDAP:
- “Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories” on page 1135
- “Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only” on page 1136
- “Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 1137
- “Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 1138
- “Managing repositories in a federated repository configuration” on page 1164

At this point, you are viewing the LDAP repository configuration page of the administrative console.
1. Enter a unique identifier for the repository in the Repository identifier field. This identifier uniquely identifies the repository within the cell, for example: LDAP1.
2. Select the type of LDAP server that is used from the Directory type list. The type of LDAP server determines the default filters that are used by WebSphere Application Server.

   IBM Tivoli Directory Server users can choose either IBM Tivoli Directory Server or SecureWay as the directory type. Use the IBM Tivoli Directory Server directory type for better performance. For a list of supported LDAP servers, see “Using specific directory servers as the LDAP server” on page 1102.
3. Enter the fully qualified host name of the primary LDAP server in the Primary host name field. You can enter either the IP address or the domain name system (DNS) name.

4. Enter the server port of the LDAP directory in the Port field. The host name and the port number represent the realm for this LDAP server in a mixed version nodes cell. If servers in different cells are communicating with each other using Lightweight Third Party Authentication (LTPA) tokens, these realms must match exactly in all the cells.

   The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

   If multiple WebSphere Application Servers are installed and configured to run in the same single sign-on domain, or if WebSphere Application Server interoperates with a previous version of WebSphere Application Server, then it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as 389 in a Version 5.x or 6.0.x configuration, and WebSphere Application Server at Version 6.1 is going to interoperate with the Version 5.x or 6.0.x server, then verify that port 389 is specified explicitly for the Version 6.1 server.
5. **Optional:** Enter the host name of the failover LDAP server in the Failover host name field. You can specify a secondary directory server to be used in the event that your primary directory server becomes unavailable. After switching to a secondary directory server, LDAP repository attempts to reconnect to the primary directory server every 15 minutes.
6. **Optional:** Enter the port of the failover LDAP server in the Port field and click **Add**. The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.
7. **Optional:** Select the type of *referral*. A referral is an entity that is used to redirect a client request to another LDAP server. A referral contains the names and locations of other objects. It is sent by the server to indicate that the information that the client requested can be found at another location, possibly at another server or several servers. The default value is ignore.

   ignore
   Referrals are ignored.

   follow Referrals are followed automatically.
8. **Optional:** Enter the bind DN name in the Bind distinguished name field, for example, cn=root. The bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information or for write operations. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed. If the LDAP server is set up to use anonymous binds, leave this field blank. If a name is not specified, the application server binds anonymously.
9. **Optional:** Enter the password that corresponds to the bind DN in the Bind password field.
10. **Optional:** Enter the property names to use to log into WebSphere Application Server in the Login properties field. This field takes multiple login properties, delimited by a semicolon (;). For example, `uid;mail`.

    All login properties are searched during login. If multiple entries or no entries are found, an exception is thrown. For example, if you specify the login properties as `uid;mail` and the login ID as Bob, the search filter searches for uid=Bob or mail=Bob. When the search returns a single entry, then authentication can proceed. Otherwise, an exception is thrown.
11. **Optional:** Select the certificate map mode in the Certificate mapping field. You can use the X.590 certificates for user authentication when LDAP is selected as the repository. The Certificate mapping field is used to indicate whether to map the X.509 certificates into an LDAP directory user by EXACT_DN or CERTIFICATE_FILTER. If EXACT_DN is selected, the DN in the certificate must exactly match the user entry in the LDAP server, including case and spaces.
12. If you select **CERTIFICATE_FILTER** in the Certificate mapping field, specify the LDAP filter for mapping attributes in the client certificate to entries in LDAP.

    If more than one LDAP entry matches the filter specification at run time, authentication fails because the result is an ambiguous match. The syntax or structure of this filter is:

`LDAP attribute=${Client certificate attribute}`

For example, `uid=${SubjectCN}`.

The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- `${UniqueKey}`
- `${PublicKey}`
- `${PublicKey}`
- `${Issuer}`
- `${NotAfter}`
- `${NotBefore}`
- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectCN}`
- `${Version}`

13. **Optional:** Select the **Require SSL communications** option if you want to use Secure Sockets Layer communications with the LDAP server.

    If you select the **Require SSL communications** option, you can select either the **Centrally managed** or **Use specific SSL alias** option.

    Centrally managed

    Enables you to specify an SSL configuration for a particular scope, such as the cell, node, server, or cluster in one location. To use the Centrally managed option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an LDAP registry, you can override the inherited SSL configuration by specifying an SSL configuration for LDAP. To specify an SSL configuration for LDAP, complete the following steps:

    a. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.

    b. Expand **Outbound >** *cell_name* **> Nodes >** *node_name* **> Servers >** *server_name* **> LDAP**.

    Use specific SSL alias

    Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu that follows the option.

    This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the following steps:

    a. Click **Security > SSL certificate and key management**.

    b. Under Configuration settings, click **Manage endpoint security configurations and trust zones >** *configuration_name*.

    c. Under Related items, click **SSL configurations**.

14. Click **OK**.

After completing these steps, your LDAP repository settings are configured.

Return to the appropriate task to complete the steps for your federated repository configuration:

- “Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories” on page 1135
- “Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only” on page 1136
- “Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 1137
- “Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 1138
- “Managing repositories in a federated repository configuration” on page 1164

Lightweight Directory Access Protocol repository configuration settings:

Use this page to configure secure access to a Lightweight Directory Access Protocol (LDAP) repository with optional failover servers.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Repository identifier:

Specifies a unique identifier for the LDAP repository. This identifier uniquely identifies the repository within the cell, for example: LDAP1.

Directory type:

Specifies the type of LDAP server to which you connect.

Expand the drop-down list to display a list of LDAP directory types.

Primary host name:

Specifies the host name of the primary LDAP server. This host name is either an IP address or a domain name service (DNS) name.

Port:

Specifies the LDAP server port.

The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

Data type:	Integer	
Default:	389	
Range:	389, which is not a Secure Sockets Layer (SSL) connection	636, which is a Secure Sockets Layer (SSL) connection

Failover host name:

Specifies the host name of the failover LDAP server.

You can specify a secondary directory server to be used in the event that your primary directory server becomes unavailable. After switching to a secondary directory server, the LDAP repository attempts to reconnect to the primary directory server every 15 minutes.

Port:

Specifies the port of the failover LDAP server.

The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

Data type:	Integer	
Range:	389, which is not a Secure Sockets Layer (SSL) connection	636, which is a Secure Sockets Layer (SSL) connection

Support referrals to other LDAP servers:

Specifies how referrals that are encountered by the LDAP server are handled.

A referral is an entity that is used to redirect a client request to another LDAP server. A referral contains the names and locations of other objects. It is sent by the server to indicate that the information that the client requested can be found at another location, possibly at another server or several servers. The default value is ignore.

Default:	ignore
Range:	**ignore** Referrals are ignored.
	follow Referrals are followed automatically.

Bind distinguished name:

Specifies the distinguished name (DN) for the application server to use when binding to the LDAP repository.

If no name is specified, the application server binds anonymously. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

Bind password:

Specifies the password for the application server to use when binding to the LDAP repository.

Login properties:

Specifies the property names to use to log into the application server.

This field takes multiple login properties, delimited by a semicolon (;). For example, `uid;mail`. All login properties are searched during login. If multiple entries or no entries are found, an exception is thrown. For

example, if you specify the login properties as `uid;mail` and the login ID as Bob, the search filter searches for uid=Bob or mail=Bob. When the search returns a single entry, then authentication can proceed. Otherwise, an exception is thrown.

Certificate mapping:

Specifies whether to map X.509 certificates into an LDAP directory by EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.

Certificate filter:

Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP repository.

If more than one LDAP entry matches the filter specification at run time, authentication fails because the result is an ambiguous match. The syntax or structure of this filter is:

```
LDAP attribute=${Client certificate attribute}
```

For example, `uid=${SubjectCN}`.

The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- `${UniqueKey}`
- `${PublicKey}`
- `${PublicKey}`
- `${Issuer}`
- `${NotAfter}`
- `${NotBefore}`
- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectCN}`
- `${Version}`

Require SSL communications:

Specifies whether secure socket communication is enabled to the LDAP server.

When enabled, the Secure Sockets Layer (SSL) settings for LDAP are used, if specified.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations, rather than spreading them across the configuration documents.

Default: Enabled
Range: Enabled or Disabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

Adding an external repository in a federated repository configuration:

Follow this task to add an external repository into a federated repository configuration.

1. If the Lightweight Directory Access Protocol (LDAP) repository that you want to add to your federated repository configuration is previously configured, select the corresponding Repository on the Repository reference panel. To access the Repository reference panel, complete the following steps:
   a. Click **Security > Secure administration, applications, and infrastructure**.
   b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
   c. Click **Add base entry to realm**.
2. Enter a distinguished name for the realm base entry in the Distinguished name that uniquely identifies... field. This base entry must uniquely identify the external repository in the realm. If multiple repositories are included in the realm, use this field to define an additional distinguished name (DN) that uniquely identifies this set of entries within the realm. For example, repositories LDAP1 and LDAP2 might both use o=ibm,c=us as the base entry in the repository. Use the DN in this field to uniquely identify this set of entries in the realm. For example: o=ibm,c=us for LDAP1 and o=ibm2,c=us for LDAP2. The specified DN in this field maps to the LDAP DN of the base entry within the repository.
3. Enter the LDAP DN of the base entry within the repository in the Distinguished name of a base entry... field. The base entry indicates the starting point for searches in this LDAP directory server. This entry and its descendents are mapped to the subtree that is identified by this unique base name entry field. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, specify the LDAP base entry as any of the following options:

   `ou=Rochester, o=IBM, c=us` or `o=IBM, c=us` or `c=us`

   In most cases, this LDAP DN is the same as the distinguished name for the realm base entry.

   If this field is left blank, then the subtree defaults to the root of the LDAP repository. Consult your LDAP administrator to determine if your LDAP repository provides support to search from the root, or create users and groups under the root without defining a suffix beforehand.

   In WebSphere Application Server, the distinguished name is normalized according to the LDAP specification. Normalization consists of removing spaces in the base distinguished name before or after commas and equal symbols. An example of a non-normalized base distinguished name is o = ibm, c = us or o=ibm, c=us. An example of a normalized base distinguished name is o=ibm,c=us.
4. If the LDAP repository that you want to add to your realm is not previously configured, complete the following steps:
   a. Click **Add Repository** on the Repository reference panel to configure the LDAP repository. See step 1 to access the Repository reference panel.
   b. Configure LDAP on the LDAP configuration panel, as described in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 1140.
   c. Select the new Repository on the Repository reference panel.
5. Click **OK**.

You have added a new or previously configured external repository into your federated repository configuration.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 1161.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify

that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.

3. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a property extension repository in a federated repository configuration:

Follow this task to configure a property extension repository to store attributes that cannot be stored in your Lightweight Directory Access Protocol (LDAP) server.

For security and business reasons, you might not want to not allow write operations to your repositories. However, applications calling the federated repository configuration might need to store additional properties for the entities. A federated repository configuration provides a *property extension repository*, which is a database regardless of the type of main profile repositories, for a property-level join configuration. For example, a company that uses an LDAP directory for its internal employees and a database for external customers and business partners might not allow write access to its LDAP and its database. The company can use the *property extension repository* in a federated repository configuration to store additional properties for the people in those repositories, excluding the user ID. When an application uses the federated repository configuration to retrieve an entry for a person, the federated repository configuration transparently joins the properties of the person that is retrieved from either the LDAP or the customer's database with the properties of the person that is retrieved from the property extension repository into a single logical person entry.

When you configure a property extension repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, then the system uses the direct access configuration.

Restriction: You cannot configure a property extension repository in a mixed-version deployment manager cell.

1. Configure the WebSphere Application Server data source. See "Configuring the WebSphere Application Server data source" on page 1158.
2. If you are adding new properties (including properties that are stored in the property extension repository) to the schema, you must do the following before you create the property extension repository.
   a. Open or create the `wimxmlextension.xml` file under the `<WAS61>\profiles\<propfile_name>\config\cells\<cell_name>\wim\model` directory.

      Attention: Make sure the editor is on the deployment manager node.

   b. Add the schema definition of the new property. The following sample `wimxmlextension.xml` file adds a new property called ibm-otherEmail to both the Person and PersonAccount entity types. This new property type is ″String″ and it is multiple-valued.

```
<sdo:datagraph xmlns:sdo="commonj.sdo"
    xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:schema>
    <wim:propertySchema nsURI="http://www.ibm.com/websphere/wim"
        dataType="String"
        multiValued="true" propertyName="ibm-otherEmail">
      <wim:applicableEntityTypeNames>Person</wim:applicableEntityTypeNames>
      <wim:applicableEntityTypeNames>PersonAccount
```

```
      </wim:applicableEntityTypeNames>
    </wim:propertySchema>
</wim:schema>
</sdo:datagraph>
```

Available data types are defined in `com.ibm.websphere.wim.SchemaConstants`. For example:

```
/**
  * Instance Class: java.lang.String
  */
 String DATA_TYPE_STRING = "String";
 /**
  * Instance Class: int
  */
 String DATA_TYPE_INT = "Int";
 /**
  * Instance Class: java.lang.Object
  */
 String DATA_TYPE_DATE = "Date";
 /**
  * Instance Class: dobjava.lang.Object
  */
 String DATA_TYPE_ANY_SIMPLE_TYPE = "AnySimpleType";
 /**
  * Instance Class: java.lang.String
  */
 String DATA_TYPE_ANY_URI = "AnyURI";
 /**
  * Instance Class: java.lang.boolean
  */
 String DATA_TYPE_BOOLEAN = "Boolean";
 /**
  * Instance Class: long
  */
 String DATA_TYPE_LONG = "Long";
 /**
  * Instance Class: double
  */
 String DATA_TYPE_DOUBLE = "Double";
 /**
  * Instance Class: short
  */
 String DATA_TYPE_SHORT = "Short";
```

c. Add the new property to the property extension repository. Before running the `setupIdMgrPropertyExtensionRepositoryTables` command, add the new properties into `<WAS61>\profiles\<propfile_name>\config\cells\<cell_name>\wim\config\wimlaproperties.xml`.

d. Follow the example inside this file to define the new property definitions. The schema file for `wimlaproperties.xml` is `wimdbproperty.xsd` and is in the same directory. It can be used for reference.

e. Run the `setupIdMgrPropertyExtensionRepositoryTables` command to create the property extension repository and to add the new properties.

3. Set up the property extension repository using wsadmin by following the procedure discussed in “Setting up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands” on page 1151; ignore the ″Before you begin″ options.

4. Configure the property extension repository by completing the following steps:

   a. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.

   b. Under User account repository, select **Federated repositories**, and click **Configure**.

   c. Click **Property extension repository**.

   d. Supply the name of the data source in the Data source name field.

   e. Select the type of database that is used for the property extension repository.

f. Supply the name of the Java database connectivity (JDBC) driver in the JDBC driver field.
   Values include:

   DB2 `COM.ibm.db2.jdbc.app.DB2Driver`

   Oracle
   `oracle.jdbc.driver.OracleDriver`

   Informix
   `com.informix.jdbc.IfxDriver`

   Microsoft SQL Server
   `com.microsoft.jdbc.sqlserver.SQLServerDriver`

   Derby `org.apache.derby.jdbc.EmbeddedDriver`

   DB2 for z/OS
   `com.ibm.db2.jcc.DB2Driver`

   DB2 for iSeries
   `com.ibm.db2.jdbc.app.DB2Driver`

g. Supply the database URL that is used to access the property extension repository with JDBC in the Database URL field. Use an alphanumeric text string that conforms to the standard JDBC URL syntax.
   Values include:

   DB2 `jdbc:db2:wim`

   Oracle
   `jdbc:oracle:thin:@<hostname>:1521:orcl`

   Derby `jdbc:derby:c:\derby\wim`

   Microsoft SQL Server
   `jdbc:microsoft:sqlserver://<hostname>:1433;databaseName=wim;selectmethod=cursor;`

   Informix
   `jdbc:informix-sqli://<hostname>:1526/wim:INFORMIXSERVER=<IFXServerName>;`

h. Supply the user name of the database administrator in the Database administrator user name field.

i. Supply the password of the database administrator in the Password field.

j. Specify the entity retrieval limit in the Entity retrieval limit field. The entity retrieval limit is the maximum number of entities that the system can retrieve from the property extension repository with a single database query. The default value is `200`.

k. Click **OK**.

After completing these steps, your federated repository configuration, which includes a property extension repository, is configured.

1. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
2. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Property extension repository settings:

Use this page to configure a property extension repository that is used to store attributes that cannot be stored in existing repositories.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Property extension repository**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Data source name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source that is used to access the property extension repository.

Default: `jdbc/wimDS`

Database type:

Specifies the type of database that is used for the property extension repository.

Default: `DB2`

JDBC driver:

Specifies the Java Database Connectivity (JDBC) driver that is used to access the entry mapping repository.

Values include:

DB2 `COM.ibm.db2.jdbc.app.DB2Driver`

Oracle
`oracle.jdbc.driver.OracleDriver`

Informix
`com.informix.jdbc.IfxDriver`

Microsoft SQL Server
`com.microsoft.jdbc.sqlserver.SQLServerDriver`

Derby `org.apache.derby.jdbc.EmbeddedDriver`

DB2 for z/OS
`com.ibm.db2.jcc.DB2Driver`

DB2 for iSeries
`com.ibm.db2.jdbc.app.DB2Driver`

Database URL:

Specifies the Web address for the property extension repository.

Values include:

DB2 `jdbc:db2:wim`

Oracle
`jdbc:oracle:thin:@<hostname>:1521:orcl`

Derby `jdbc:derby:c:\derby\wim`

Microsoft SQL Server

```
jdbc:microsoft:sqlserver://<hostname>:1433;databaseName=wim;selectmethod=cursor;
```

Informix

```
jdbc:informix-sqli://<hostname>:1526/wim:INFORMIXSERVER=<IFXServerName>;
```

Database administrator user name:

Specifies the user name of the database administrator that is used to access the property extension repository.

Password:

Specifies the password that is used to enable the database administrator to access the property extension repository.

Entity retrieval limit:

Specifies the maximum number of entities that the system can retrieve from the property extension repository with a single database query.

Data type:	Integer
Default:	200

Setting up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands:

You can set up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands.

If you are setting up an entry mapping repository, begin with the steps described in “Configuring an entry mapping repository in a federated repository configuration” on page 1158.

If you are setting up a property extension repository, begin with the steps described in “Configuring a property extension repository in a federated repository configuration” on page 1147.

Three types of repositories are currently supported: DB2 repository, property extension repository, and entry mapping repository. When a repository is created, use the appropriate wsadmin command to define the database schema and to populate the database property definitions.

1. Create an empty entry mapping repository as shown in the following examples:
   a. For DB2, open a DB2 command window or command center and enter the following:

      ```
      db2 create database <name> using codeset UTF-8 territory US
      ```

   b. Enter the following database tuning commands:

      ```
      db2 update database configuration for <name> using applheapsz 1024
      db2 update database configuration for <name> using stmtheap 4096
      db2 update database configuration for <name> using app_ctl_heap_sz 2048
      db2 update database configuration for <name> using locklist 1024
      db2 update database configuration for <name> using indexrec RESTART
      db2 update database configuration for <name> using logfilsiz 1000
      db2 update database configuration for <name> using logprimary 12
      db2 update database configuration for <name> using logsecond 10
      db2 update database configuration for <name> using sortheap 2048
      db2set DB2_RR_TO_RS=yes
      ```

   c. **Optional:** For Informix databases using `dbaccess`, enter the following command:

      ```
      CREATE DATABASE <name> WITH BUFFERED LOG
      ```

d. **Optional:** For Oracle databases, the database should already exist during Oracle installation (for example, `orcl`).

2. Run the `setupIdMgrDBTables` command by doing the following:
   a. Start WebSphere Application Server.
   b. Open a command window and go to the `<WAS>/Profiles/<PROFILE_NAME>bin` directory.
   c. Start wsadmin.
   d. Type the necessary commands as described below.

The `setupIdMgrDBTables` command can be used to:

- Specify the arguments on the command line.
- Specify the arguments in a file.

For the commands below, the `-file` option enables you to specify a file in which some or all of the parameters are specified. To use the `-file` argument on the command line, enter the full path to the file. Parameters in the file must be specified in `key=value` pairs and each must be on its own line. If a parameter is specified on both the command line and in the file, the value on the command line takes precedence.

Note: If an argument is not properly specified on the command line or in the file, a message is returned which states that the argument was not properly specified. This might mean that the argument was not specified at all or was required for a given configuration but was not specified.

If the argument was not specified at all, check that the parameter is specified on the command line or in the file, and that it is properly spelled and has matching case.

If the argument was required for a given configuration but was not specified, it is possible that a value is not required solely by the command but is required for the type of database and configuration you are setting.

For example, if you set the `dn`, `wasAdminId`, or `wasAdminPassword` parameters, you must also specify the `dbDriver` parameter. Additionally, if the `dn`, `wasAdminId` or `wasAdminPassword` parameters are specified, and the `databaseType` is not a Cloudscape 10 Version 1 database, then the `dbAdminId` and `dbAdminPassword` parameters must also be specified.

The setupIdMgrDBTables command:

The setupIdMgrDBTables command sets up the database, which includes creating and populating the tables in the database. Required arguments are prefixed by a double start (**). Arguments are case-sensitive, both through the command line and the file.

Parameters:

****schemaLocation (String)**
The location of the `<WAS>/etc/wim/setup` directory.

dbPropXML (String)
The location of database repository property definition XML file.

****databaseType (String)**
The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

dn (String)
The default organization uniqueName to replace. For example: `o=yourco`. If it is not set, `o=Default Organization` is used.

wasAdminId (String)
The WebSphere Application Server admin user ID. The ID should be a short name, not a uniqueName. For example: wasadmin. After creation, the uniqueName is `uid=wasadmin,<defaultOrg>`.

wasAdminPassword (String)
The WebSphere Application Server admin user password. If `wasAdminId` is set, then this parameter is mandatory.

saltLength (Integer)
The salt length of the randomly generated salt for password hashing.

encryptionKey (String)
The password encryption key. Set the password encryption key to match the encryption key in the wimconfig.xml file for the repository. If the encryption key is not set, the default is used.

derbySystemHome (String)
The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

The deleteIdMgrDBTables command:

The deleteIdMgrDBTables command deletes the tables in the database.

Parameters:

****schemaLocation (String)**
The location of the `<WAS>/etc/wim/setup` directory.

****databaseType (String)**
The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

The setupIdMgrPropertyExtensionRepositoryTables command:

The setupIdMgrPropertyExtensionRepositoryTables command sets up the property extension repository, which includes creating and populating the tables in the database.

Parameters:

****schemaLocation (String)**
The location of the `<WAS>/etc/wim/setup` directory.

laPropXML (String)
The location of the property extension repository definition XML file.

****databaseType (String)**
The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**
The database URL for direct access mode. For example: jdbc:db2:wim.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

The deleteIdMgrPropertyExtensionRepositoryTables command:

The deleteIdMgrPropertyExtensionRepositoryTables command deletes the tables in the property extension database.

Parameters:

****schemaLocation (String)**
The location of the `<WAS>/etc/wim/setup` directory.

****databaseType (String)**
The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

The setupIdMgrEntryMappingRepositoryTables command:

The setupIdMgrEntryMappingRepositoryTables command sets up the entry mapping repository, which includes creating and populating the tables of the repository.

Parameters:

****schemaLocation (String)**
The location of the `<WAS>/etc/wim/setup` directory.

****databaseType (String)**
The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

The deleteIdMgrEntryMappingRepositoryTables command:

The deleteIdMgrEntryMappingRepositoryTables command deletes the tables in the entry mapping repository.

Parameters:

****schemaLocation (String)**
The location of the `<WAS>/etc/wim/setup` directory.

****databaseType (String)**
The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**
The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)
The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)
The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)
The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)
The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)
Specifies whether to report SQL errors while setting up databases.

file (String)
The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a `key=value` pair. Each pair must be on a separate line.

Sample command line usage:

To set up a database using the command line, enter the following:

```
$AdminTask setupIdMgrDBTables {-schemaLocation "C:\WAS7\etc\wim\setup" -dbPropXML
"C:\WAS7\etc\wim\setup\wimdbproperties.xml" -databaseType db2
-dbURL jdbc:db2:wim -dbAdminId db2admin
-dbDriver COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd
-reportSqlError true}
```

To delete database tables using the command line, enter the following:

```
$AdminTask deleteIdMgrDBTables {-schemaLocation "C:\WAS7\etc\wim\setup"
-databaseType db2  -dbURL jdbc:db2:wim -dbAdminId db2admin
-dbDriver COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd
-reportSqlError true}
```

To set up a property extension repository using the command line, enter the following:

```
$AdminTask setupIdMgrPropertyExtensionRepositoryTables {-schemaLocation
"C:\WAS7\etc\wim\setup"
-laPropXML "C:\WAS7\etc\wim\setup\wimlaproperties.xml" -databaseType db2
-dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver COM.ibm.db2.jdbc.app.DB2Driver
-dbAdminPassword db2adminPwd -reportSqlError true}
```

To delete a property extension repository using the command line, enter the following:

```
$AdminTask deleteIdMgrPropertyExtensionRepositoryTables {-schemaLocation "C:\WAS7\etc\wim\setup "
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver
COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

To set up an entry mapping repository using the command line, enter the following:

```
$AdminTask setupIdMgrEntryMappingRepositoryTables {-schemaLocation "C:\WAS7\etc\wim\setup"
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver
COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

To delete an entry mapping repository using the command line, enter the following:

```
$AdminTask deleteIdMgrEntryMappingRepositoryTables {-schemaLocation "C:\WAS7\etc\wim\setup"
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver
COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

Sample CLI Usage using -file option:

To set up a database with the `-file` option using the example `params.txt` file below, enter the following:

```
$AdminTask setupIdMgrDBTables {–file C:\params.txt -dbPropXML
"C:\OverrideDBPropParam\wimdbproperties.xml"}
```

Params.txt

```
schemaLocation=C:\WAS7\etc\wim\setup
dbPropXML=C:\Program Files\IBM\WebSphere\AppServer\profiles\default
\config\cells\mycell\wim\config\wimdbproperties.xml
laPropXML=C:\Program Files\IBM\WebSphere\AppServer\profiles\default
\config\cells\mycell\wim\config\wimlaproperties.xml
databaseType=db2
dbURL=jdbc:db2:wim
dbDriver=COM.ibm.db2.jdbc.app.DB2Driver
reportSqlError=true
dn=o=db.com
dbAdminId=db2admin
dbAdminPassword=dbPassword
wasAdminId=wasadmin
wasAdminPassword=wasadmin1
```

To set up a database with the `-file` option using a file only, enter the following:

```
$AdminTask setupIdMgrDBTables {-file C:\params.txt}
```

Note: The use of a file only works if `-file` is the only parameter specified on the command line. If other parameters are specified then the file is completely ignored, and only the parameters on the command line are used to execute the command.

Configuring the WebSphere Application Server data source:

This section describes how to configure the data source service in WebSphere Application Server.

1. Start the WebSphere Application Server administrative console.
2. Click **Security -> Secure administration, applications and infrastructure**.
3. On the Configuration panel, expand Java Authentication and Authorization Service and click **J2C authentication data**.
4. Click **New** and enter the Alias, User ID and Password.
5. Click **Ok**.
6. Click **Resources -> JDBC -> JDBC Providers**.
7. In the Scope section, choose the **Node level**.
8. Click **New** to create a new JDBC driver.
9. Select the Database type, Provider type, Implementation type and Name.
10. Click **Next** and configure the database class path. Click **Next**.
11. On the Summary page, click **Finish**.
12. In the Additional properties section, click **Data sources**.
13. Click **New** to create a new data source. Enter the Data source name and the JNDI name, and choose the authentication alias from the drop-down list in Component-managed authentication alias. The JNDI name should match the datasourceName value set in `wimconfig.xml`. By default, it is `jdbc/wimDS`.

    Note: For Cloudscape 10 Version 1 embedded databases, leave the Component-managed authentication alias field set to `NONE`.
14. Click **Next**.
15. Enter the Database name and deselect **Use this data source in container managed persistence (CMP)**. Click **Next**.
16. Under Component-managed authentication alias, select the authentication alias previously created. Click **Test Connection**. The message should indicate that the connection is successful. Ignore any warnings, and then click **Next**.
17. On the Summary page, click **Finish**.
18. Save the configurations, and restart WebSphere Application Server.

Configuring an entry mapping repository in a federated repository configuration:

Follow this task to configure an entry mapping repository that is used to store data for managing profiles on multiple repositories.

An *entry-level join* means that the federated repository configuration uses multiple repositories simultaneously and recognizes the entries in the different repositories as entries representing distinct entities. For example, a company might have a Lightweight Directory Access Protocol (LDAP) directory that contains entries for its employees and a database that contains entries for business partners and customers. By configuring an entry mapping repository, a federated repository configuration can use both the LDAP and the database at the same time. The federated repository configuration hierarchy and constraints for identifiers provide the aggregated namespace for both of those repositories and prevent identifiers from colliding.

When you configure an entry mapping repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, then the system uses the direct access configuration.

Restriction: You cannot configure an entry mapping repository in a mixed-version deployment manager cell.

1. Configure the WebSphere Application Server data source. See "Configuring the WebSphere Application Server data source" on page 1158.
2. Set up the entry mapping repository using wsadmin. See "Setting up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands" on page 1151.
3. Configure the entry mapping repository into the federated repository by doing the following:
   a. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
   b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
   c. Click **Entry mapping repository**.
   d. Supply the name of the data source in the Data source name field.
   e. Select the type of database that is used for the property extension repository.
   f. Supply the name of the Java database connectivity (JDBC) driver in the JDBC driver field.

      Values include:

      DB2 `COM.ibm.db2.jdbc.app.DB2Driver`

      Oracle
      `oracle.jdbc.driver.OracleDriver`

      Informix
      `com.informix.jdbc.IfxDriver`

      Microsoft SQL Server
      `com.microsoft.jdbc.sqlserver.SQLServerDriver`

      Derby `org.apache.derby.jdbc.EmbeddedDriver`

      DB2 for z/OS
      `com.ibm.db2.jcc.DB2Driver`

      DB2 for iSeries
      `com.ibm.db2.jdbc.app.DB2Driver`
   g. Supply the database URL that is used to access the property extension repository with JDBC in the Database URL field. Use an alphanumeric text string that conforms to the standard JDBC URL syntax.

      Values include:

      DB2 `jdbc:db2:wim`

      Oracle
      `jdbc:oracle:thin:@<hostname>:1521:orcl`

      Derby `jdbc:derby:c:\derby\wim`

      Microsoft SQL Server
      `jdbc:microsoft:sqlserver://<hostname>:1433;databaseName=wim;selectmethod=cursor;`

      Informix
      `jdbc:informix-sqli://<hostname>:1526/wim:INFORMIXSERVER=<IFXServerName>;`
   h. Supply the user name of the database administrator in the Database administrator user name field.
   i. Supply the password of the database administrator in the Password field.

j. Click **OK**.

After completing these steps, your federated repository configuration, which includes an entry mapping repository, is configured.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Entry mapping repository settings:

Use this page to configure an entry mapping repository that is used to store data for managing profiles on multiple repositories.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Entry mapping repository**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Data source name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source that is used to access the entry mapping repository.

Default: `jdbc/wimDS`

Database type:

Specifies the type of database that is used to access the entry mapping repository.

Default: `DB2`

JDBC driver:

Specifies the Java Database Connectivity (JDBC) driver that is used to access the entry mapping repository.

Values include:

DB2 `COM.ibm.db2.jdbc.app.DB2Driver`

Oracle
oracle.jdbc.driver.OracleDriver

Informix
com.informix.jdbc.IfxDriver

Microsoft SQL Server
com.microsoft.jdbc.sqlserver.SQLServerDriver

Derby org.apache.derby.jdbc.EmbeddedDriver

DB2 for z/OS
com.ibm.db2.jcc.DB2Driver

DB2 for iSeries
com.ibm.db2.jdbc.app.DB2Driver

Database URL:

Specifies the Web address for the entry mapping repository.

Values include:

DB2 jdbc:db2:wim

Oracle
jdbc:oracle:thin:@<*hostname*>:1521:orcl

Derby jdbc:derby:c:\derby\wim

Microsoft SQL Server
jdbc:microsoft:sqlserver://<*hostname*>:1433;databaseName=wim;selectmethod=cursor;

Informix
jdbc:informix-sqli://<*hostname*>:1526/wim:INFORMIXSERVER=<*IFXServerName*>;

Database administrator user name:

Specifies the user name of the database administrator that is used to access the entry mapping repository.

Password:

Specifies the password that is used to enable the database administrator to access the entry mapping repository.

Configuring supported entity types in a federated repository configuration:

Follow this task to configure supported entity types for user and group management.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The supported entity types are Group, OrgContainer, and PersonAccount. A Group entity represents a simple collection of entities that might not have any relational context. An OrgContainer entity represents an organization, such as a company or an enterprise, a subsidiary, or an organizational unit, such as a division, a location, or a department. A PersonAccount entity represents a human being. You cannot add or delete the supported entity types, because these types are predefined.

The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Click **Supported entity types** to view a list of predefined entity types.
4. Click the name of a predefined entity type to change its configuration.
5. Supply the distinguished name of a base entry in the repository in the Base entry for the default parent field. This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.
6. Supply the relative distinguished name (RDN) properties for the specified entity type in the Relative Distinguished Name properties field. Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

   The following list outlines known requirements and limitations that apply to specific Lightweight Directory Access Protocol (LDAP) servers:

   Using Microsoft Active Directory as the LDAP server

   - Unless you modify the LDAP schema to use `uid`, you must specify `cn` in the Relative Distinguished Name (RDN) properties field for the PersonAccount entity type.
   - Secure Sockets Layer communications must be enabled to create users with passwords. To select the **Require SSL communications** option, see the topic “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 1140.
   - Typically the value of `user` is specified as the value in the Object classes field for the PersonAccount entity type and the value of `group` is specified as the value in the Object classes field for the Group entity type.

   Using a Lotus Domino Enterprise Server as the LDAP server

   - Typically, the value of `cn` is specified in the Relative Distinguished Name (RDN) properties field for the PersonAccount entity type. The value of `uid` is also acceptable.
   - Typically, both `inetOrgPerson` and `dominoPerson` are used as values in the Object classes field for the PersonAccount entity type.

   Using Sun ONE Directory Server as the LDAP server

   - Typically, `groupOfUniqueNames` is specified as the value in the Object classes field for the Group entity type.
7. Click **OK**.

After completing these steps, your federated repository configuration, which uses supported entity types, is configured.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Supported entity types collection:

Use this page to list entity types that are supported by the member repositories or to select an entity type to view or change its configuration properties.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Supported entity types**.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Related reference

"Lightweight Directory Access Protocol entity types settings" on page 1171
Use this page to configure Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories.

Entity type:

Specifies the entity type name.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Supported entity types settings:

Use this page to configure entity types that are supported by the member repositories.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Supported entity types**.
4. Click the name of a configured entity type to view or change its configuration.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Entity type:

Specifies the name of the entity type.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Managing repositories in a federated repository configuration:

Follow this topic to manage repositories in a federated repository configuration.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**. Repositories that are configured in the system are listed in the collection panel. This list includes repositories that are configured using the federated repository functionality as well as repositories that are created using wsadmin commands described in the topic “Commands for the IdMgrRepositoryConfig group of the AdminTask object” on page 1533.
4. **Optional:** Click **Add** to configure a new external repository. The Lightweight Directory Access Protocol (LDAP) repository configuration settings are described in detail in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 1140.

   Restriction: You cannot add a database repository using the administrative console. This repository configuration is supported by using wsadmin commands only.
5. **Optional:** Click **Delete** to delete a repository that you specified previously using the administrative console or wsadmin commands.

   Restriction: You cannot delete the built-in, file-based repository from the collection panel.
6. **Optional:** Select one of the LDAP repository identifier entries to view or update an external repository that is configured in the system previously. The steps to configure LDAP settings are described in detail in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 1140.

   Restriction: While database repositories that are configured in the system are listed in the collection panel, you cannot update a database repository using the administrative console. Updates to a database repository are supported by using wsadmin commands only.
7. Click **OK**.

After completing these steps, the collection panel under Managing repositories reflects a current list of repositories that are configured in your system.

1. To add one or more external repositories that are listed on this collection panel into the realm, see “Managing the realm in a federated repository configuration” on page 1129.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Manage repositories collection:

Use this page to list repositories that are configured in the system or to select a repository to view or change its configuration properties. You can add or delete external repositories.

To view this administrative console page, complete the following steps:
1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository type:

Specifies the repository type, such as File or LDAP.

Repository reference settings:

Use this page to configure a repository reference. A repository reference is a single repository that contains a set of identity entries that are referenced by a base entry into the directory information tree.

To view this administrative console page, complete the following steps:
1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Click **Add base entry to realm**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Repository:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Expand the drop-down list to display a list of previously defined repository identifiers.

Distinguished name that uniquely identifies this set of entries in the realm:

Specifies the distinguished name (DN) that uniquely identifies this set of entries in the realm.

If multiple repositories are included in the realm, it is necessary to define an additional distinguished name that uniquely identifies this set of entries within the realm.

Distinguished name of a base entry in this repository:

Specifies the Lightweight Directory Access Protocol (LDAP) distinguished name (DN) of the base entry within the repository. The entry and its descendents are mapped to the subtree that is identified by the unique base name entry field.

If this field is left blank, then the subtree defaults to the root of the LDAP repository.

Increasing the performance of the federated repository configuration:

Follow this page to manage the realm in a federated repository configuration.

The settings that are available on the Performance panel are independent options that pertain specifically to the federated repositories functionality. These options do not affect your entire WebSphere Application Server configuration.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories** > *repository_name*.
4. Under Additional properties, click **Performance**.
5. **Optional:** Select the **Limit search time** option and enter the maximum number of milliseconds that the Application Server can use to search through your Lightweight Directory Access Protocol (LDAP) entries.
6. **Optional:** Select the **Limit search returns** option and enter the maximum number of entries to return that match the search criteria.
7. **Optional:** Select the **Use connection pooling** option to specify whether the Application Server can store separate connections to the LDAP server for reuse.
8. **Optional:** Select the **Enable context pool** option to specify whether multiple applications can use the same connection to the LDAP server. If you select the option, specify the initial, preferred, and maximum number of entries that can use the same connection. The **Enable context pool** option can be enabled either in conjunction with the **Use connection pool** option or separately. If this option is disabled, a new connection is created for each context. You can also select the **Context pool times out** option and specify the number of seconds after which the entries in the context pool expire.
9. **Optional:** Select the **Cache the attributes** option and specify the maximum number of search attribute entries. This option enables WebSphere Application Server to save the LDAP entries so that it can search the entries locally rather than making multiple calls to the LDAP server. Click the **Cache times out** option that is associated with the **Cache the attributes** option to specify the maximum number of seconds that the Application Server can save these entries.

10. **Optional:** Select the **Cache the search results** option and specify the maximum number of search result entries. This option enables WebSphere Application Server to save the results of a search inquiry instead of making multiple calls to the LDAP server to search and retrieve the results of that search. Click the **Cache times out** option that is associated with the **Cache the search results** option to specify the maximum number of seconds that the Application Server can save the results.

These options are available to potentially increase the performance of your federated repositories configuration. However, the any increase in performance is dependant upon your specific configuration.

Lightweight Directory Access Protocol performance settings:

Use this page to minimize impacts to performance by adding opened connections and contexts to internally maintained pools and reusing them. Also minimize performance impacts by maintaining internal caches of retrieved data.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Performance**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Limit search time:

Specifies the timeout value in milliseconds for a Lightweight Directory Access Protocol (LDAP) server to respond before stopping a request.

Data type:	Integer
Units:	Milliseconds
Default:	0
Range:	Equal to or greater than 0. A value of 0 specifies that no search time limit exists.

Limit search returns:

Specifies the maximum number of entries that are returned in a search result.

Data type:	Integer
Units:	Entries
Default:	0
Range:	Equal to or greater than 0. A value of 0 specifies that no search return limit exists.

Use connection pooling:

Specifies whether to utilize the connection pooling function, which is provided in the Software Development Kit (SDK).

Connection pooling is maintained by the Java run time. It is configured by system properties.

Default: Disabled
Range: Enabled or Disabled

Enable context pool:

Specifies whether context pooling is enabled to the LDAP server. To improve performance, use the context pool in combination with connection pooling.

Default: Enabled
Range: Enabled or Disabled

Initial size:

Specifies the number of context instances in the pool when the pool is initially created by the LDAP repository.

Data type: Integer
Default: 1
Range: 1 to 50

Preferred size:

Specifies the preferred number of context instances that the context pool maintains. Both in-use and idle context instances contribute to this number.

Data type: Integer
Default: 3
Range: 0 to 100

Maximum size:

Specifies the maximum number of context instances that can be maintained concurrently by the context pool. Both in-use and idle context instances contribute to this number.

When the pool size reaches the maximum size, no new context instances can be created for a new request. The new request is blocked until a context instance is released or removed. The request periodically checks for context instances that are available in the pool. A request for a pooled context instance uses an existing pooled and idle context instance or a newly created pooled context instance.

A maximum pool size of 0 indicates that the context pool can maintain an infinite number of context instances.

Data type: Integer
Default: 0

Context pool times out:

Specifies the number of seconds for the context pool to time out and remove idle context instances.

A timeout value of 0 indicates that the context pool does not time out context instances.

Data type: Integer
Default: 0

Cache the attributes:

Specifies whether to cache the attributes that are returned from the LDAP server.

Default: Enabled
Range: Enabled or Disabled

Cache size:

Specifies the maximum size of the cache.

Data type: Integer
Default: 4000
Range: Equal to or greater than 100

Cache times out:

Specifies the maximum number of seconds that the cached search results can stay in the cache.

A timeout value of 0 indicates that the cached search results stay in the cache until update operations are made.

Data type: Integer
Units: Seconds
Default: 1200
Range: Equal to or greater than 0

Cache the search results:

Specifies whether to cache the search results that are returned from the LDAP server.

Default: Enabled
Range: Enabled or Disabled

Cache size:

Specifies the maximum size of the cache.

Data type: Integer
Default: 2000
Range: Equal to or greater than 100

Cache times out:

Specifies the maximum number of seconds that the cached search results can stay in the cache.

A timeout value of 0 indicates that the cached search results stay in the cache until update operations are made.

Data type:	Integer
Units:	Seconds
Default:	600
Range:	Equal to or greater than 0

Configuring Lightweight Directory Access Protocol entity types in a federated repository configuration:

Follow this task to configure Lightweight Directory Access Protocol (LDAP) entity types in a federated repository configuration.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

   Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
5. Under Additional properties, click **LDAP entity types**.
6. View the entity types that are supported by the member repositories, or select an entity type to view or change its configuration properties.
7. Supply the object classes that are mapped to this entity type in the Object classes field. LDAP entries that contain one or more of the object classes belong to this entity type.
8. Supply the search bases that are used to search this entity type. The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

   `o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

   In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

   Delimit multiple search bases with a semicolon (;). For example:

   `ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us`
9. Supply the LDAP search filter that is used to search this entity type.

   For example, use `(objectclass=ePerson)` to search for users or `(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)` to search for groups in an external LDAP repository.

   If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter. For information on RDN properties, see “Configuring supported entity types in a federated repository configuration” on page 1161.

After completing these steps, LDAP entity types are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.

3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Lightweight Directory Access Protocol entity types collection:

Use this page to list Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories or to select an LDAP entity type to view or change its configuration properties.

To view this administrative console page, complete the following steps:
1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type name.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

Lightweight Directory Access Protocol entity types settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories.

To view this administrative console page, complete the following steps:
1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.
6. Select an entity type to view or change its configuration properties.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

Search bases:

Specifies the search bases that are used to search this entity type.

The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

`o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

Delimit multiple search bases with a semicolon (;). For example:

```
ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us
```

Search filter:

Specifies the LDAP search filter that is used to search this entity type.

For example, use `(objectclass=ePerson)` to search for users or `(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)` to search for groups in an external LDAP repository.

If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter.

Configuring group attribute definition settings in a federated repository configuration:

Follow this task to configure group definition settings in a federated repository configuration.

Because group attribute definition settings apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see “Managing repositories in a federated repository configuration” on page 1164.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

   Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
5. Under Additional properties, click **Group attribute definition**.
6. Supply the name of the group membership attribute in the Name of group membership attribute field. Only one membership attribute can be defined for each LDAP repository.

   Every LDAP entry should have this attribute to indicate the groups to which this entry belongs. For example, memberOf is the name of the membership attribute that is used in Active Directory. The group membership attribute contains values that reference groups to which this entry belongs. If UserA belongs to GroupA, then the value of the memberOf attribute of UserA should contain the distinguished name of GroupA.

If your LDAP server does not support the group membership attribute, then do not specify this attribute. The LDAP repository can look up groups by searching the group member attributes, though the performance might be slower.

7. Select the scope of the group membership attribute. The default value is Direct.

   Direct The membership attribute contains direct groups only. Direct groups are the groups that contain the member. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct group of User1, but Group1 is not a direct group of User1.

   Nested
   The membership attribute contains both direct groups and nested groups.

   All The membership attribute contains direct groups, nested groups, and dynamic members.

After completing these steps, group attribute definition settings are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Group attribute definition settings:

Use this page to specify the name of the group membership attribute. Every Lightweight Directory Access Protocol (LDAP) entry includes this attribute to indicate the group to which this entry belongs.

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

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name of group membership attribute:

Specifies the name of the group membership attribute. Only one membership attribute can be defined for each Lightweight Directory Access Protocol (LDAP) repository.

Every LDAP entry should have this attribute to indicate the groups to which this entry belongs. For example, memberOf is the name of the membership attribute that is used in Active Directory. The group membership attribute contains values that reference groups to which this entry belongs. If UserA belongs to GroupA, then the value of the memberOf attribute of UserA should contain the distinguished name of GroupA.

If your LDAP server does not support the group membership attribute, then do not specify this attribute. The LDAP repository can look up groups by searching the group member attributes, though the performance might be slower.

Scope of group membership attribute:

Specifies the scope of the group membership attribute.

Default: Direct

Range:

Direct The membership attribute contains direct groups only. Direct groups are the groups that contain the member. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct group of User1, but Group1 is not a direct group of User1.

Nested The membership attribute contains both direct groups and nested groups.

All The membership attribute contains direct groups, nested groups, and dynamic members.

Configuring member attributes in a federated repository configuration:

Follow this task to configure member attributes in a federated repository configuration.

Because member attributes apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see “Managing repositories in a federated repository configuration” on page 1164.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

   Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.
7. Click **New** to specify a new member attribute or **Delete** to remove a preconfigured member attribute.
8. Accept the default, or supply the name of the member attribute in the Name of member attribute field. For example, member and uniqueMember are two commonly used names of member attributes.

   The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.
9. Supply the object class of the group that uses this member attribute in the Object class field. If this field is not defined, this member attribute applies to all group object classes.
10. Select the scope of the member attribute. The default value is Direct.

    Direct The member attribute contains direct members only. Direct members are members that are

directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.

Nested
The member attribute contains both direct members and nested members.

All The member attribute contains direct members, nested members, and dynamic members.

After completing these steps, member attributes are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in "Enabling security for the realm" on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Member attributes collection:

Use this page to list Lightweight Directory Access Protocol (LDAP) member attributes or to select a member attribute to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name:

Specifies the name of the member attribute in LDAP. For example, member and uniqueMember are two commonly used names of member attributes.

The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.

Scope:

Specifies the scope of the member attribute.

Default: Direct

Range:

Direct The member attribute contains direct members only. Direct members are members that are directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.

Nested The member attribute contains both direct members and nested members.

All The member attribute contains direct members, nested members, and dynamic members.

Object class:

Specifies the object class of the group that uses this member attribute. If this field is not defined, this member attribute applies to all group object classes.

Member attributes settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.
7. Click **New** to specify a new member attribute.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name of member attribute:

Specifies the name of the member attribute in LDAP. For example, member and uniqueMember are two commonly used names of member attributes.

The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.

Object class:

Specifies the object class of the group that uses this member attribute. If this field is not defined, this member attribute applies to all group object classes.

Scope:

Specifies the scope of the member attribute.

Default: Direct

Range:

Direct The member attribute contains direct members only. Direct members are members that are directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.

Nested The member attribute contains both direct members and nested members.

All The member attribute contains direct members, nested members, and dynamic members.

Configuring dynamic member attributes in a federated repository configuration:

Follow this task to configure dynamic member attributes in a federated repository configuration.

Because dynamic member attributes apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see "Managing repositories in a federated repository configuration" on page 1164.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

   Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Dynamic member attributes**.
7. Click **New** to specify a new dynamic member attribute or **Delete** to remove a preconfigured dynamic member attribute.
8. Accept the default, or supply the name of the dynamic member attribute in the Name of dynamic member attribute field. The name of the dynamic member attribute defines the filter for dynamic group members in LDAP, for example, memberURL is the name of a commonly used dynamic member attribute.

   If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

   A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

   ```
   ldap:///<base DN of search> ? ? <scope of search> ? <searchfilter>
   ```

   The following is an example of the LDAP URL that defines all entries that are under `o=Acme` with the `objectclass=person`:

   ```
   ldap:///o=Acme,c=US??sub?objectclass=person
   ```

9. Supply the object class of the group that contains the dynamic member attribute in the Dynamic object class field, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.

After completing these steps, dynamic member attributes are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 1071. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Dynamic member attributes collection:

Use this page to manage Lightweight Directory Access Protocol (LDAP) dynamic member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Dynamic member attributes**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name:

Specifies the name of the attribute that defines the filter for dynamic group members in LDAP. For example, memberURL is the name of a commonly used dynamic member attribute.

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search>??<scope of search>?<searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under `o=Acme` with the `objectclass=person`:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

Object class:

Specifies the object class of the group that contains this dynamic member attribute, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.

Dynamic member attributes settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) dynamic member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Dynamic member attributes**.
7. Click **New** to specify a new dynamic member attribute.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name of dynamic member attribute:

Specifies the name of the attribute that defines the filter for dynamic group members in LDAP. For example, memberURL is the name of a commonly used dynamic member attribute.

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search>??<scope of search>?<searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under `o=Acme` with the `objectclass=person`:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

Dynamic object class:

Specifies the object class of the group that contains this dynamic member attribute, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.

Local operating system registries

With the registry implementation for the local operating system, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

Lightweight Directory Access Protocol (LDAP) is a centralized registry. Most local operating system registries are not centralized registries.

WebSphere Application Server provides implementations for the Windows local accounts registry and domain registry, as well as implementations for the Linux, Solaris, and AIX user accounts registries. Windows Active Directory is supported through the LDAP user registry implementation discussed later.

Note: For an Active Directory (domain controller), the three group scopes are Domain Local Group, Global Group, and Universal Group. For an Active Directory (Domain Controller), the two group types are Security and Distribution.

When a group is created, the default value is Global and the default type is Security. With Windows NT domain registry support for Windows 2000 and 2003 domain controllers, WebSphere Application Server only supports Global groups that are the Security type. It is recommended that you use the Active Directory registry support rather than a Windows NT domain registry if you use Windows 2000 and 2003 domain controllers because the Active Directory supports all group scopes and types. The Active Directory also supports a nested group that is not support by Windows NT domain registry. The Active Directory is a centralized control registry.

WebSphere Application Server does not have to install the member of the domain because it can be installed on any machine on any platform. Note that the Windows NT domain native call returns the support group only without an error.

Do not use a local operating system registry in a WebSphere Application Server environment where application servers are dispersed across more than one machine because each machine has its own user registry.

The Windows domain registry and Network Information Services (NIS) are exceptions. Both the Windows domain registry and Network Information Services (NIS) are centralized registries. The Windows domain registry is supported by WebSphere Application Server; however, NIS is not supported.

As mentioned previously, the access IDs taken from the user registry are used during authorization checks. Because these IDs are typically unique identifiers, they vary from machine to machine, even if the exact users and passwords exist on each machine.

Web client certificate authentication is not currently supported when using the local operating system user registry. However, Java client certificate authentication does function with a local operating user registry. Java client certificate authentication maps the first attribute of the certificate domain name to the user ID in the user registry.

Even though Java client certificates function correctly, the following error displays in the `SystemOut.log` file:

```
CWSCJ0337E: The mapCertificate method is not supported
```

The error is intended for Web client certificates; however, it also displays for Java client certificates. Ignore this error for Java client certificates.

Required privileges

The user that is running the WebSphere Application Server process requires enough operating system privilege to call the Windows systems application programming interface (API) for authenticating and obtaining user and group information from the Windows operating system. This user logs into the machine,

or if running as a service, is the Log On As user. Depending on the machine and whether the machine is a standalone machine or a machine that is part of a domain or is the domain controller, the access requirements vary.

- For a standalone machine, the user:
  - Is a member of the administrative group.
  - Has the Act as part of the operating system privilege.
  - Has the Log on as a service privilege, if the server is run as a service.
- For a machine that is a member of a domain, only a domain user can start the server process and:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Act as part of the operating system privilege in the Local security policy on the local machine.
  - Has the Log on as a service privilege on the local machine, if the server is run as a service.

    The user is a domain user and not a local user, which implies that when a machine is part of a domain, only a domain user can start the server.
- For a domain controller machine, the user:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Log on as a service privilege on the domain controller, if the server is run as a service.

If the user running the server does not have the required privilege, you might see one of the following exception messages in the log files:

- `A required privilege is not held by the client.`
- `Access is denied.`

Domain and local user registries

When WebSphere Application Server is started, the security run-time initialization process dynamically attempts to determine if the local machine is a member of a Windows domain. If the machine is part of a domain then by default both the local registry users or groups and the domain registry users or groups can be used for authentication and authorization purposes with the domain registry taking precedence. The list of users and groups that is presented during the security role mapping includes users and groups from both the local user registry and the domain user registry. The users and groups can be distinguished by the associated host names.

WebSphere Application Server does not support trusted domains.

If the machine is not a member of a Windows system domain, the user registry local to that machine is used.

Using both the domain user registry and the local operating system registry

When the machine that hosts the WebSphere Application Server process is a member of a domain, both the local and the domain user registries are used by default. The following section describes more on this topic and recommends some best practices to avoid unfavorable consequences.

Note: Although this section does not directly describe z/OS considerations, you should be aware that overall security operations are affected by how well you set up these registries.

- **Best practices**

  In general, if the local and the domain registries do not contain common users or groups, it is simpler to administer and it eliminates unfavorable side effects. If possible, give users and groups access to unique security roles, including the server ID and administrative roles. In this situation, select the users and groups from either the local user registry or the domain user registry to map to the roles.

In cases where the same users or groups exist in both the local user registry and the domain user registry, it is recommended that at least the server ID and the users and groups that are mapped to the administrative roles be unique in the registries and exist only in the domain.

If a common set of users exists, set a different password to make sure that the appropriate user is authenticated.

- **How it works**

  When a machine is part of a domain, the domain user registry takes precedence over the local user registry. For example, when a user logs into the system, the domain user registry tries to authenticate the user first. If authentication fails, the local user registry is used. When a user or a group is mapped to a role, the user and group information is first obtained from the domain user registry. In case of failure, the local user registry is tried.

  However, when a fully qualified user or a group name, one with an attached domain or host name, is mapped to a role, only that user registry is used to get the information. Use the administrative console or scripts to get the fully qualified user and group names, which is the recommended way to map users and groups to roles.

  Tip: A user, `Bob`, on one machine in the local OS user registry, for example, is not the same as the user `Bob` on another machine in the domain user registry, for example, because the unique ID of `Bob`, which is the security identifier [SID] in this case, is different in different user registries.

- **Examples**

  The `MyMachine` machine is part of the `MyDomain` domain. The `MyMachine` machine contains the following users and groups:
  - MyMachine\user2
  - MyMachine\user3
  - MyMachine\group2

  The `MyDomain` domain contains the following users and groups:
  - MyDomain\user1
  - MyDomain\user2
  - MyDomain\group1
  - MyDomain\group2

  Here are some scenarios that assume the previous set of users and groups:
  1. When `user2` logs into the system, the domain user registry is used for authentication. If the authentication fails because the password is different, for example, the local user registry is used.
  2. If the `MyMachine\user2` user is mapped to a role, only the `user2` user in `MyMachine` machine has access. Thus, if the `user2` password is the same on both the local and the domain user registries, the `user2` user cannot access the resource because the `user2` user is always authenticated using the domain user registry. If both user registries have common users, it is recommended that you have different passwords.
  3. If the `group2` group is mapped to a role, only the users who are members of the `MyDomain\group2` group can access the resource because `group2` information is first obtained from the domain user registry.
  4. If the `MyMachine\group2` group is mapped to a role, only the users who are members of the `MyMachine\group2` group can access the resource. A specific group is mapped to the role (`MyMachine\group2` instead of just `group2`).
  5. Use either the `user3` user or the `MyMachine\user3` user to map to a role because the `user3` user is unique as it exists in one user registry only.

  Authorizing with the domain user registry first can cause problems if a user exists in both the domain and local user registries with the same password. Role-based authorization can fail in this situation because the user is first authenticated within the domain user registry. This authentication produces a unique domain security ID that is used in WebSphere Application Server during the authorization check. However, the local user registry is used for role assignment. The domain security ID does not match the unique security ID that is associated with the role. To avoid this problem, map security roles to domain users instead of local users.

Using either the local or the domain user registry. If you want to access users and groups from either the local or the domain user registry, instead of both, set the com.ibm.websphere.registry.UseRegistry property. This property can be set to either `local`or `domain`. When this property is set to `local`(case insensitive) only the local user registry is used. When this property is set to domain, (case insensitive) only the domain user registry is used.

Set this property by completing the following steps to access the **Custom Properties** panel in the administrative console:

1. Click **Security > Secure administration, applications, and infrastructure**
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Local operating system**, and click **Configure**.
3. Under Additional properties, click **Custom properties**.

You can also use wsadmin to configure this property. When the property is set, the privilege requirement for the user who is running the product process does not change. For example, if this property is set to `local`, the user that is running the process requires the same privilege, as if the property was not set.

Using Linux and Solaris system user registries Linux Solaris

For WebSphere Application Server local operating system registry to work on the Linux and Solaris platforms, a shadow password file must exist. The shadow password file is named `shadow` and is located in the `/etc` directory. If the shadow password file does not exist, an error occurs after enabling administrative security and configuring the registry as local operating system.

To create the shadow file, run the **`pwconv`** command (with no parameters). This command creates an `/etc/shadow` file from the `/etc/passwd` file. After creating the shadow file, you can enable local operating system security successfully.

Standalone Lightweight Directory Access Protocol registries

A Standalone Lightweight Directory Access Protocol (LDAP) registry performs authentication using an LDAP binding.

WebSphere Application Server security provides and supports the implementation of most major LDAP directory servers, which can act as the repository for user and group information. These LDAP servers are called by the product processes for authenticating a user and other security-related tasks. For example, the servers are used to retrieve user or group information. This support is provided by using different user and group filters to obtain the user and group information. These filters have default values that you can modify to fit your needs. The custom LDAP feature enables you to use any other LDAP server, which is not in the product-supported list of LDAP servers, for its user registry by using the appropriate filters.

To use LDAP as the user registry, you need to know a administrative user name that is defined in the registry, the server host and port, the base distinguished name (DN) and, if necessary, the bind DN and the bind password. You can choose any valid user in the registry that is searchable and have administrative privileges. In some LDAP servers, the administrative users are not searchable and cannot be used, for example, `cn=root` in SecureWay. This user is referred to as WebSphere Application Server security server ID, server ID, or server user ID in the documentation. Being a server ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log into the administrative console after security is turned on. You can use other users to log in if those users are part of the administrative roles.

When security is enabled in the product, the primary administrative user name and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. It is important to choose an ID and password that do not expire or change often. If the product server user ID or password need to change in the registry, make sure that the changes are performed when all the product servers are up and running.

When the changes are done in the registry, use the steps that are described in “Configuring Lightweight Directory Access Protocol user registries” on page 1088. Change the ID, password, and other configuration information, save, stop, and restart all the servers so that the new ID or password is used by the product. If any problems occur starting the product when security is enabled, disable security before the server can start up. To avoid these problems, make sure that any changes in this panel are validated in the Secure administration, applications, and infrastructure panel. When the server is up, you can change the ID, password, and other configuration information and then enable security.

You can use the custom Lightweight Directory Access Protocol (LDAP) feature to support any LDAP server by setting up the correct configuration. However, support is not extended to these custom LDAP servers because many configuration possibilities exist.

The users and groups and security role mapping information is used by the configured authorization engine to perform access control decisions.

Dynamic groups and nested group support:

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

Dynamic groups contain a group name and membership criteria:

- The group membership information is as current as the information on the user object.
- There is no need to manually maintain members on the group object.
- Dynamic groups are designed so an application does not need a large amount of information from the directory to find out if someone is a member of a group.

Nested groups enable the creation of hierarchical relationships that are used to define inherited group membership. A nested group is defined as a child group entry whose distinguished name (DN) is referenced by a parent group entry attribute.

You only need to assign a larger parent group if all nested groups share the same privilege. Assigning a role to a single parent group simplifies the run-time authorization table.

Dynamic groups and nested group support for the IBM Tivoli Directory Server:

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

WebSphere Application Server supports all Lightweight Directory Access Protocol (LDAP) dynamic and nested groups when using IBM Tivoli Directory Server. This function is enabled by default by taking advantage of a new feature in IBM Tivoli Directory Server. IBM Tivoli Directory Server uses the ibm-allGroups forward-reference group attribute that automatically calculates all the group memberships including dynamic and recursive memberships for a user. Security directly locates a user group membership from a user object rather than indirectly search all the groups to match group members.

For more information, see “Configuring dynamic and nested group support for the IBM Tivoli Directory Server” on page 1106.

Dynamic and nested group support for the SunONE or iPlanet Directory Server:

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

The SunONE or iPlanet Directory Server uses two grouping mechanisms:

Groups
Entries that name other entries as a list of members or as a filter for members.

Roles Entries that name other entries as a list of members or as a filter for members. Additional functionality is provided by generating the nsrole attribute on each role member.

Three types of roles are available:

Filtered roles
Depends upon the attributes that are contained in each entry. Entries are members, if they match a specified Lightweight Directory Access Protocol (LDAP) filter. This role is equivalent to a dynamic group.

Nested roles
Creates roles that contain other roles. This role is equivalent to a nested group.

Managed roles
Explicitly assigns a role to member entries. This role is equivalent to a static group.

Refer to "Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server" on page 1106 for more information.

Security failover among multiple LDAP servers:

WebSphere Application Server security can be configured to attempt failovers between multiple Lightweight Directory Access Protocol (LDAP) hosts.

If the current active LDAP server is unavailable, WebSphere Application Server security attempts a failover to the first available LDAP host in the specified host list. The multiple LDAP servers can be replicas of the same master LDAP server, or they can be any LDAP host with the same schema, which contain data that is imported from the same LDAP Data Interchange Format (LDIF) file.

Whenever a failover occurs, WebSphere Application Server security always uses the first available LDAP server in the specified host list. For example, if there are four LDAP servers configured in the order of L1, L2, L3, and L4, L1 is treated as the primary LDAP server. The preference of connection is from L1 to L4. If, for example, WebSphere Application Server security is currently connected to L4, and failover or reconnection is necessary, WebSphere Application Server security first attempts to connect to L1, L2, and then L3 in that order until the connection is successful.

The current LDAP host name is logged in message CWSCJ0419I in the WebSphere Application Server log file, `SystemOut.log`. If you want to reconnect to the primary LDAP host, run the WebSphere Application Server MBean method, resetLDAPBindInfo, with `null,null` as the input.

To configure LDAP failover among multiple LDAP hosts, you must use wsadmin or ConfigService to include the backup LDAP host, which does not have a number limitation. The LDAP host that is displayed in the administrative console is the primary LDAP host, and is the first item listed in the LDAP host list in `security.xml`.

The WebSphere Application Server security realm name defaults to the primary LDAP host name that is displayed in the administrative console. It includes a trailing colon and a port number (if one exists). However, the custom property, com.ibm.websphere.security.ldap.logicRealm, can be added to override the default security realm name. Use the logicRealm name to configure each cell to have its own LDAP host for interoperability and backward compatibility, and to provide flexibility for adding or removing the LDAP host dynamically. If migrating from a previous installation, the new logicRealm name does not take effect until administrative security is enabled again. To be compatible with a previous release that does not support logic realm, the logicRealm name has to be the same as that used by the previous installation (the LDAP host name, including a trailing colon and port number).

The following example shows how to use wsadmin to add a backup LDAP host for failover:

```
proc LDAPAdd {args} {
 global AdminConfig AdminControl ldapServer ldapPort
 set ldapServer [lindex $args 0]
 set ldapPort [lindex $args 1]
     global ldapUserRegistryId
 if {[catch {$AdminConfig list LDAPUserRegistry} result]} {
  puts stdout "\$AdminConfig list LDAPUserRegistry caught an exception $result\n"
  return
 } else {
  if {$result != {}} {
   set ldapUserRegistryId [lindex $result 0]
  } else {
   return;
  }
 }
        set secMbean [$AdminControl queryNames type=SecurityAdmin,*]
        set Attrs2 [list [list hosts [list [list [list host $ldapServer]
                   [list port $ldapPort]]]]]
        $AdminConfig modify $ldapUserRegistryId $Attrs2
        $AdminConfig save
}
```

Federated repositories

Federated repositories enable you to use multiple repositories with WebSphere Application Server. These repositories, which can be file-based repositories, LDAP repositories, or a sub-tree of an LDAP repository, are defined and theoretically combined under a single realm. All of the user repositories that are configured under the federated repository functionality are invisible to WebSphere Application Server.

When you use the federated repositories functionality, all of the configured repositories, which you specify as part of the federated repository configuration, become active. It is recommended that the user ID, and the distinguished name (DN) for an LDAP repository, be unique in multiple user repositories that are configured under the same federated repository configuration. For example, there might be three different repositories that are configured for the federated repositories configuration: Repository A, Repository B, and Repository C. When user1 logs in, the federated repository adapter searches each of the repositories for all of the occurrences of that user. If multiple instances of that user are found in the combined repositories, an error message displays.

In addition, the federated repositories functionality in WebSphere Application Server supports the logical joining of entries across multiple user repositories when the Application Server searches and retrieves entries from the repositories. For example, when an application calls for a sorted list of people whose age is greater than twenty, WebSphere Application searches all of the repositories in the federated repositories configuration. The results are combined and sorted before the Application Server returns the results to the application.

Unlike the local operating system, standalone LDAP registry, or custom registry options, federated repositories provide user and group management with read and write capabilities. When you configure federated repositories, you can use one of the following methods to add, create, and delete users and groups:

Important: If you configure multiple repositories under the federated repositories realm, you must also configure supported entity types and specify a base entry for the default parent. The base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management. See “Configuring supported entity types in a federated repository configuration” on page 1161 for details.

- Use the user management application programming interfaces (API). For more information, refer to articles under ″Developing with virtual member manager″ in this information center.
- Use the administrative console. To manage users and groups within the administrative console, click **Users and Groups > Manage Users** or **Users and Groups > Manage Groups**. For information on

user and group management, click the Help link that displays in the upper right corner of the window. From the left navigation pane, click **Users and Groups**.

- Use the wsadmin commands. For more information, see “Commands for the WIMManagementCommands group of the AdminTask object” on page 1630.

If you do not configure the federated repositories functionality or do not enable federated repositories as the active repository, you cannot use the user management capabilities that are associated with federated repositories. You can configure an LDAP server as the active user registry and configure the same LDAP server under federated repositories, but not select federated repositories as the active user repository. With this scenario, authentication takes place using the LDAP server, and you can use the user management functionality for the LDAP server that is available for federated repositories.

The following table compares the federated repository functionality that is available in WebSphere Application Server Version 6.1 with the registry functionality that remains unchanged from previous versions of the Application Server.

Table 9. Federated repositories versus user registry implementations

Federated repositories	User registry
Supports multiple types of repositories such as file-based, LDAP, database, and custom. In WebSphere Application Server Version 6.1, file-based and LDAP repositories are supported by the administrative console. However, the federated repositories functionality does not support local operating system implementations. For database and custom repositories, you can use the wsadmin command-line interface or the configuration application programming interfaces (API).	Supports multiple types of registries such as the local operating system, a standalone LDAP registry, and a standalone custom registry.
Supports multiple repositories in a realm within a cell.	Supports one registry only in a realm within a cell.
Provides read and write capabilities for the repositories that are defined in the federated repository configuration.	Provides read only capability for the registries.
Provides account and password policy support as defined by the registry type. However, this support is not provided by the federated repository functionality.	Provides account and password policy support as defined by the registry type.
Supports identity profiles.	Does not support identity profiles.
Uses the custom UserRegistry implementation.	Uses the custom UserRegistry implementation.

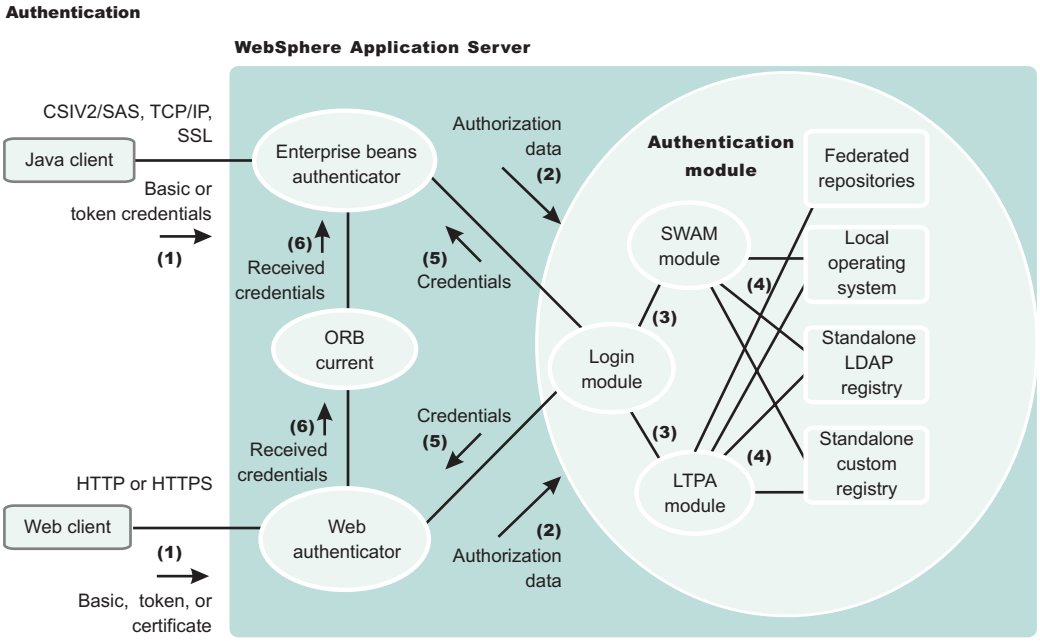
Authentication mechanisms

An *authentication mechanism* defines rules about security information, such as whether a credential is forwardable to another Java process, and the format of how security information is stored in both credentials and tokens.

Authentication is the process of establishing whether a client is who or what it claims to be in a particular context. A client can be either an end user, a machine, or an application. An authentication mechanism in WebSphere Application Server typically collaborates closely with a *user registry*. The user registry is the user and groups account repository that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a *credential*, which is an internal product representation of a successfully authenticated client user. Not all credentials are created equally. The abilities of the credential are determined by the configured authentication mechanism.

WebSphere Application Server provides two authentication mechanisms: Lightweight Third Party Authentication (LTPA) and Simple WebSphere Authentication Mechanism (SWAM). You configure LTPA, which is the default authentication mechanism, in the administrative console by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.

SWAM is deprecated in Version 6.1. SWAM does not provide authenticated communication between different servers. To use SWAM instead of LTPA, select the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel. However, if you select the **Use SWAM-no authenticated communication between servers** option, the other information on the Authentication mechanisms and expiration panel will be ignored.



Authentication process

The figure demonstrates the authentication process. Authentication is required for enterprise bean clients and Web clients when they access protected resources. Enterprise bean clients, like a servlet or other enterprise beans or a pure client, send the authentication information to a Web application server using one of the following protocols: V6.0.x

- Common Secure Interoperability Version 2 (CSIv2)
- Secure Authentication Service (SAS)

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Web clients use the HTTP or HTTPS protocol to send the authentication information, as shown in the previous figure.

The authentication information can be basic authentication (user ID and password), a credential token (in the case of Lightweight Third Party Authentication (LTPA)), or a client certificate. The Web authentication is performed by the Web Authentication module.

You can configure Web authentication for a Web client by using the administrative console. Click **Security > Secure administration and applications**. Under Authentication, expand **Web security** and click **General settings**. The following options exist for Web authentication:

Authenticate only when the URI is protected
Specifies that the Web client can retrieve an authenticated identity only when it accesses a protected Uniform Resource Identifier (URI). WebSphere Application Server challenges the Web

client to provide authentication data when the Web client accesses a URI that is protected by a J2EE role. This default option is also available in previous versions of WebSphere Application Server.

Use available authentication data when an unprotected URI is accessed
Specifies that the Web client is authorized to call the `getRemoteUser`, `isUserInRole`, and `getUserPrincipal` methods; retrieves an authenticated identity from either a protected or an unprotected URI. Although the authentication data is not used when you access an unprotected URI, the authentication data is retained for future use. This option is available when you select the **Authentication only when the URI is protected** check box.

Authenticate when any URI is accessed
Specifies that the Web client must provide authentication data regardless of whether the URI is protected.

Default to basic authentication when certificate authentication for the HTTPS client fails.
Specifies that WebSphere Application Server challenges the Web client for a user ID and password when the required HTTPS client certificate authentication fails.

V6.0.x The enterprise bean authentication is performed by the Enterprise JavaBean (EJB) authentication module, which resides in the CSIv2 and SAS layer.

The authentication module is implemented using the Java Authentication and Authorization Service (JAAS) login module. The Web authenticator and the EJB authenticator pass the authentication data to the login module (2), which can use any of the following mechanisms to authenticate the data:

- LTPA
- Simple WebSphere Authentication Mechanism (SWAM)

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. LTPA is the default authentication mechanism.

The authentication module uses the registry that is configured on the system to perform the authentication (4). Four types of registries are supported:

- Federated repositories
- Local operating system
- Standalone Lightweight Directory Access Protocol (LDAP) registry
- Standalone custom registry

External registry implementation following the registry interface that is specified by IBM can replace either the local operating system or the LDAP registry.

The login module creates a JAAS subject after authentication and stores the credential that is derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or to the enterprise beans authenticator (5).

The Web authenticator and the enterprise beans authenticator store the received credentials in the Object Request Broker (ORB) current for the authorization service to use in performing further access control checks. If the credentials are forwardable, they are sent to other application servers.

Portlet URL security

WebSphere Application Server enables direct access to portlet Uniform Resource Locators (URLs), just like servlets. This section describes security considerations when accessing portlets using URLs.

For security purposes, portlets are treated similar to servlets. Most portlet security uses the underlying servlet security mechanism. However, portlet security information resides in the `portlet.xml` file, while the

servlet and JavaServer Pages files reside in the web.xml file. Also, when you make access decisions for portlets, the security information, if any, in the web.xml file is combined with the security information in the portlet.xml file.

Portlet security must support both programmatic security, that is isUserInRole, and declarative security. The programmatic security is exactly the same as for servlets. However, for portlets, the isUserInRole method uses the information from the security-role-ref element in portlet.xml. The other two methods used by programmatic security, getRemoteUser and getUserPrincipal, behave the same way as they do when accessing a servlet. Both of these methods return the authenticated user information accessing the portlet.

The declarative security aspect of the portlets is defined by the security-constraint information in the portlet.xml file. This is similar to the security-constraint information used for the servlets in the web.xml file with the following differences:

- The auth-constraint element, which lists the names of the roles that can access the resources, does not exist in the portlet.xml file. The portlet.xml file contains only the user-data-constraint element, which indicates what type of transport layer security (HTTP or HTTPS) is required to access the portlet.
- The security-constraint information in the portlet.xml file contains the portlet-collection element, while the web.xml file contains the web-resource-collection element. The portlet-collection element contains only a list of simple portlet names, while the web-resource-collection contains the url-patterns as well as the HTTP methods that need protection.

The portlet container does not deal with the user authentication directly. For example, it does not prompt you to collect the credential information. The portlet container must, instead, use the underlying servlet container for the user authentication mechanism. As a result, there is no auth-constraint element in the security-constraint information in the portlet.xml file.

In WebSphere Application Server, when a portlet is accessed using a URL, the user authentication is processed based on the security-constraint information for that portlet in the web.xml file. This implies that to authenticate a user for a portlet, the web.xml file must contain the security-constraint information for that portlet with the relevant auth-constraints contained in it. If a corresponding auth-constraint for the portlet does not exist in the web.xml file, it indicates that the portlet is not required to have authentication. In this case, unauthenticated access is permitted just like a URL pattern for a servlet that does not contain any auth-constraints in the web.xml file. An auth-constraint for a portlet can be specified directly by using the portlet name in the url-pattern element, or indirectly by a url-pattern that implies the portlet.

Note: You cannot have a servlet or JSP with the same name as a portlet for WebSphere Application Server security to work with portlet.

The following examples demonstrate how the security-constraint information contained in the portlet.xml and web.xml files in a portlet application are used to make security decisions for portlets. The security-role-ref element, which is used for isUserInRole calls, is not discussed here because it is used the same way for servlets.

In the examples below (unless otherwise noted), there are four portlets (MyPortlet1, MyPortlet2, MyPortlet3, MyPortlet4) defined in portlet.xml. The portlets are secured by combining the information, if any, in the web.xml file when they are accessed directly through URLs.

All of the examples show the contents of the web.xml and portlet.xml files. Use the correct tools when creating these deployment descriptor files as you normally would when assembling a portlet application.

Example 1: The web.xml file does not contain any security-constraint data

In the following example, the security-constraint information is contained in portlet.xml:

```
<security-constraint id="SecurityConstraint_1">
         <web-resource-collection id="WebResourceCollection_1">
            <web-resource-name>Protected Area</web-resource-name>
            <url-pattern>/*</url-pattern>
            </web-resource-collection>
         <auth-constraint id="AuthConstraint_1">
            <role-name>Manager</role-name>
          </auth-constraint>
      </security-constraint>
```

In this example, when you access anything under MyPortlet1 and MyPortlet3, and these portlets are accessed using the unsecured HTTP protocol, you are redirected through the secure HTTPS protocol. The transport-guarantee is set to use secure connections. For MyPortlet2 and MyPortlet4, unsecured (HTTP) access is permitted because the transport-guarantee is not set. There is no corresponding security-constraint information for all four portlets in the `web.xml` file. Therefore, all of the portlets can be accessed without any user authentication and role authorization. The only security involved in this instance is the transport-layer security using Secure Sockets Layer (SSL) for MyPortlet1 and MyPortlet3.

The following table lists the security constraints that are applicable to the individual portlets.

URL	Transport Protection	User Authentication	Role Based Authorization
/MyPortlet1/*	HTTPS	None	None
/MyPortlet2/*	None	None	None
/MyPortlet3/*	HTTPS	None	None
/MyPortlet4/*	None	None	None

Example 2: The `web.xml` file contains portlet specific security-constraint data

In the following example, the security-constraint information that corresponds to the portlet is contained in `web.xml`. The `portlet.xml` file is the same as that shown in the previous example.

```
<security-constraint id="SecurityConstraint_1">
    <web-resource-collection id="WebResourceCollection_1">
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/MyPortlet1/*</url-pattern>
      <url-pattern>/MyPortlet2/*</url-pattern>
      </web-resource-collection>
    <auth-constraint id="AuthConstraint_1">
      <role-name>Employee</role-name>
    </auth-constraint>
</security-constraint>
```

The security-constraint information contained in the `web.xml` file in this example indicates that the user authentication must be performed when accessing anything under the MyPortlet1 and MyPortlet2 portlets. When you attempt to access these portlets directly using URLs, and there is no authentication information available, you are prompted to enter their credentials. After you are authenticated, the authorization check is performed to see if you are listed in the Employee role. The user/group to role mapping is assigned during the portlet application deployment. In the `web.xml` file listed above, note the following:

- Because the `web.xml` file uses url-pattern, the portlet names have been modified slightly. MyPortlet1 is now /MyPortlet1/*, which indicates that everything under the MyPortlet1 URL is protected. This matches the information in the `portlet.xml` file because the security runtime code converts the portlet-name element in the `portlet.xml` file to url-pattern (for example, MyPortlet1 to /MyPortlet1/*), even for the transport-guarantee.
- The http-method element in the `web.xml` file is not used in the example because all HTTP methods must be protected.

The following table lists the new security constraints that are applicable to the individual portlets.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	Yes	Yes (Employee)
MyPortlet2/*	None	Yes	Yes (Employee)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	None	None

Example 3: The `web.xml` file contains generic security-constraint data implying all portlets.

In the following example, the security-constraint information is contained in the `web.xml` file that corresponds to the portlet. The `portlet.xml` file is the same as that shown in the first example.

```
<security-constraint id="SecurityConstraint_1">
    <web-resource-collection id="WebResourceCollection_1">
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/*</url-pattern>
      </web-resource-collection>
    <auth-constraint id="AuthConstraint_1">
      <role-name>Manager</role-name>
    </auth-constraint>
    </security-constraint>
```

In this example, /* implies that all resources that do not contain their own explicit security-constraints should be protected by the Manager role as per the URL pattern matching rules. Because the `portlet.xml` file contains explicit security-constraint information for MyPortlet1 and MyPortlet3, these two portlets are not protected by the Manager role, only by the HTTPS transport. Because the `portlet.xml` file cannot contain the auth-constraint information, any portlets that contain security-contraints in it are rendered unprotected when an implying URL (`/*` for example) is listed in the `web.xml` file because of the URL matching rules.

In the case above, both MyPortlet1 and MyPortlet3 can be accessed without user authentication. However, because MyPortlet2 and MyPortlet4 do not have security-constraints in the `portlet.xml` file, the /* pattern is used to match these portlets and are protected by the Manager role, which requires user authentication.

The following table lists the new security constraints that are applicable to the individual portlets with this setup.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	None	None
MyPortlet2/*	None	Yes	Yes (Manager)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	Yes	Yes (Manager)

If in the example above, if you must also protect a portlet contained in the `portlet.xml` file (for example, MyPortlet1), the `web.xml` file should contain an explicit security-constraint entry in addition to `/*` as shown in the following example:

```
<security-constraint id="SecurityConstraint_1">
         <web-resource-collection id="WebResourceCollection_1">
            <web-resource-name>Protected Area</web-resource-name>
            <url-pattern>/*</url-pattern>
            </web-resource-collection>
         <auth-constraint id="AuthConstraint_1">
            <role-name>Manager</role-name>
          </auth-constraint>
```

```
</security-constraint>
<security-constraint id="SecurityConstraint_2">
   <web-resource-collection id="WebResourceCollection_2">
      <web-resource-name>Protection for MyPortlet1</web-resource-name>
      <url-pattern>/MyPortlet1/*</url-pattern>
      </web-resource-collection>
   <auth-constraint id="AuthConstraint_1">
      <role-name>Manager</role-name>
    </auth-constraint>
</security-constraint>
```

In this case, MyPortlet1 is protected by the Manager role and requires authentication. The data-constraint of `CONFIDENTIAL` is also applied to it because the information in the `web.xml` file and the `portlet.xml` file are combined. Because MyPortlet3 is not explicitly listed in the `web.xml` file, it is still not protected by the Manager role and does not require user authentication.

The following table shows the effect of this change.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	Yes	Yes (Manager)
MyPortlet2/*	None	Yes	Yes (Manager)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	Yes	Yes (Manager)

Lightweight Third Party Authentication

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. LTPA supports forwardable credentials and single sign-on (SSO). LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

Application servers distributed in multiple nodes and cells can securely communicate using the LTPA protocol. It also provides the single sign-on (SSO) feature wherein a user is required to authenticate only once in a domain name system (DNS) domain and can access resources in other WebSphere Application Server cells without getting prompted. The realm names on each system in the DNS domain are case sensitive and must match identically.

Windows For local OS, the realm name is the domain name, if a domain is in use or the realm name is the machine name.

For the Lightweight Directory Access Protocol (LDAP), the realm name is the `host:port` value of the LDAP server.

The LTPA protocol uses cryptographic keys to encrypt and decrypt user data that passes between the servers. These keys must be shared between the different cells for the resources in one cell to access resources in other cells, assuming that all the cells involved use the same LDAP or custom registry.

When using LTPA, a token is created with the user information and an expiration time and is signed by the keys. The LTPA token is time sensitive. All product servers that participate in a protection domain must have their time, date, and time zone synchronized. If not, LTPA tokens appear prematurely expired and cause authentication or validation failures.

This token passes to other servers, in the same cell or in a different cell through cookies, for Web resources when SSO is enabled, or through the authentication protocol layer for enterprise beans.

If the receiving servers share the same keys as the originating server, the token can be decrypted to obtain the user information, which then is validated to make sure that it has not expired and that the user information in the token is valid in its registry. On successful validation, the resources in the receiving servers are accessible after the authorization check.

All of the WebSphere Application Server processes in a cell (deployment manager, nodes, application servers) share the same set of keys. If key sharing is required between different cells, export them from one cell and import them to the other. For security purposes, the exported keys are encrypted with a user-defined password. This same password is needed when importing the keys into another cell.

WebSphere Application Server supports the LTPA and the Simple WebSphere Authentication Mechanism (SWAM) protocols.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

When security is enabled during profile creation time, LTPA is configured by default.

LTPA requires that the configured user registry be a centrally shared repository such as LDAP or a Windows domain-type registry so that users and groups are the same, regardless of the machine.

The following table summarizes the authentication mechanism capabilities and user registries with which LTPA can work.

	Forwardable credentials	SSO	Local OS user registry	LDAP user registry	Custom user registry
SWAM	No	No	Yes	Yes	Yes
LTPA	Yes	Yes	Yes	Yes	Yes

Lightweight Third Party Authentication key sets and key set groups:

Key set groups contain lists of key sets and Lightweight Third Party Authentication (LTPA) key generation schedules. Each key set contains key references to keys in key stores. To generate keys automatically, each key set must be a member of a key set group.

The keys for some key configurations must be generated together. The LTPA key pair is referenced in one key set while the secret or private key is in a separate key set. When the key set group is created, the two key sets are added as members of the key set group. Key set group settings determine whether the keys for both key sets are generated together automatically or manually.

The key set group contains the following attributes:

- Member key sets
- Choice of either manual or automatic key generation in the member key sets
- Schedule for automatically generating keys

Trust associations

Trust association enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials that are passed by the proxy server.

Demand for such an integrated configuration has become more compelling, especially when a single product cannot meet all of the customer needs or when migration is not a viable solution. This article provides a conceptual background behind the approach.

In this setup, WebSphere Application Server is used as a back-end server to further exploit its fine-grained access control. The reverse proxy server passes the HTTP request to WebSphere Application Server that includes the credentials of the authenticated user. WebSphere Application Server then uses these credentials to authorize the request.

Trust association model

The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests that are received from a reverse proxy server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full trust to the proxy server and the proxy server applies its authentication policies on every Web request that is dispatched to WebSphere Application Server. This trust is validated by the interceptors that reside in the product environment for every request received. The method of validation is agreed upon by the proxy server and the interceptor.

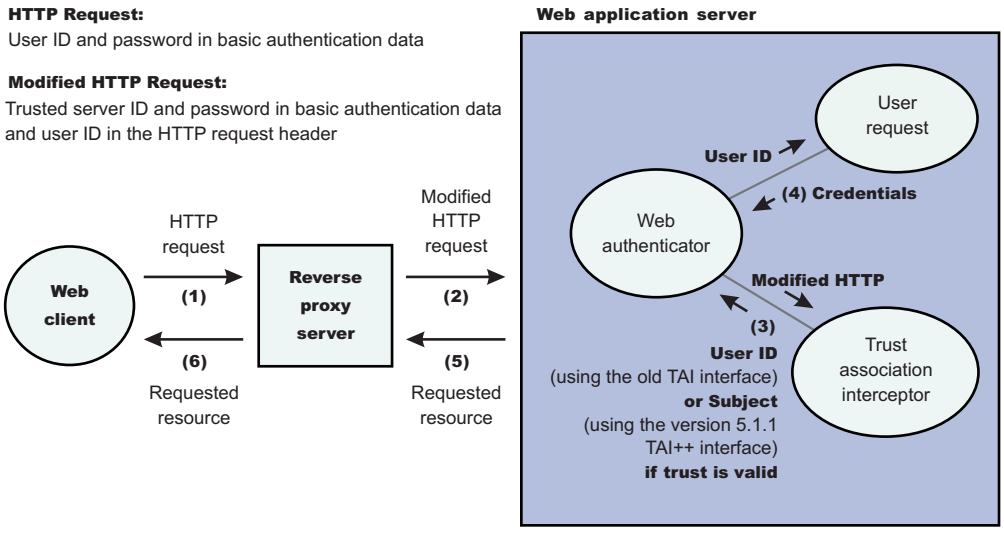
Running in trust association mode does not prohibit WebSphere Application Server from accepting requests that did not pass through the proxy server. In this case, no interceptor is needed for validating trust. It is possible, however, to configure WebSphere Application Server to strictly require that all HTTP requests go through a reverse proxy server. In this case, all requests that do not come from a proxy server are immediately denied by WebSphere Application Server.

WebSphere Application Server supports the following trust association interceptor (TAI) interfaces:

com.ibm.ws.security.web.WebSealTrustAssociationInterceptor
: This Tivoli TAI interceptor that implements the WebSphere Application Server TAI interface is provided to support WebSEAL Version 4.1. If you plan to use WebSEAL 5.1 or a later version of WebSEAL, it is recommended that you migrate to use the new com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus interceptor; which implements the new com.ibm.wsspi.security.tai.TrustAssociationInterceptor interface.

com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus
: This TAI interceptor implementation that implements the new WebSphere Application Server interface supports WebSphere Application Server Version 5.1.1 and later. The interface supports WebSEAL Version 5.1 and later, but does not support WebSEAL Version 4.1. For an explanation of security attribute propagation, see “Security attribute propagation” on page 1206

Trust association model



IBM WebSphere Application Server: WebSEAL Integration

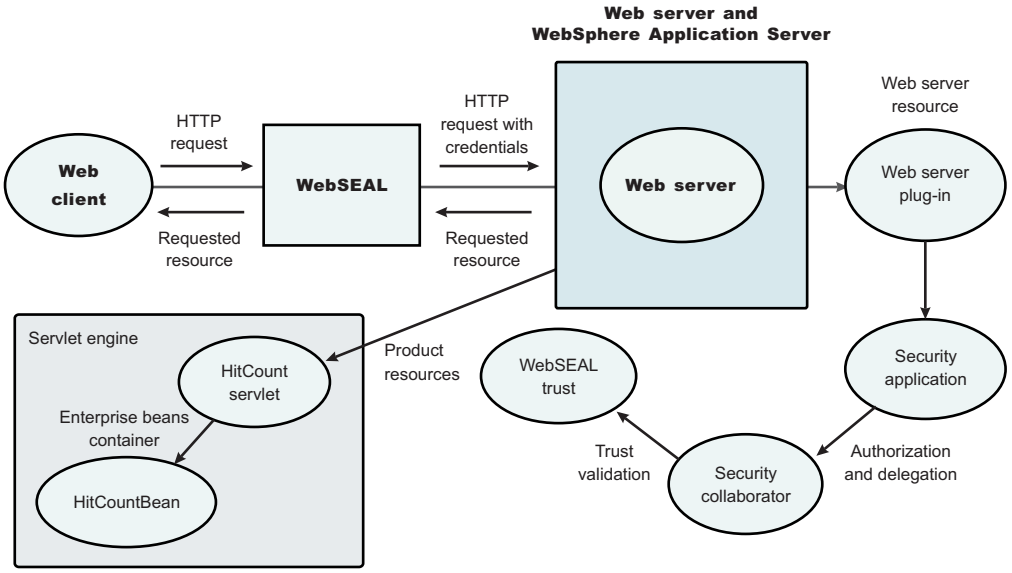
The integration of WebSEAL and WebSphere Application Server security is achieved by placing the WebSEAL server at the front-end as a reverse proxy server. From a WebSEAL management perspective, a junction is created with WebSEAL on one end, and the product Web server on the other end. A junction is a logical connection that is created to establish a path from the WebSEAL server to another server.

In this setup, a request for Web resources that are stored in a protected domain of the product is submitted to the WebSEAL server where it is authenticated against the WebSEAL security realm. If the requesting user has access to the junction, the request is transmitted to the WebSphere Application Server HTTP server through the junction, and then to the application server.

Meanwhile, WebSphere Application Server validates every request that comes through the junction to ensure that the source is a trusted party. This process is referenced as *validating the trust* and it is performed by a WebSEAL product-designated interceptor. If the validation is successful, WebSphere Application Server authorizes the request by checking whether the client user has the required permissions to access the Web resource. If so, the Web resource is delivered to the WebSEAL server through the Web server, which then gives the resource to the client user.

WebSEAL server

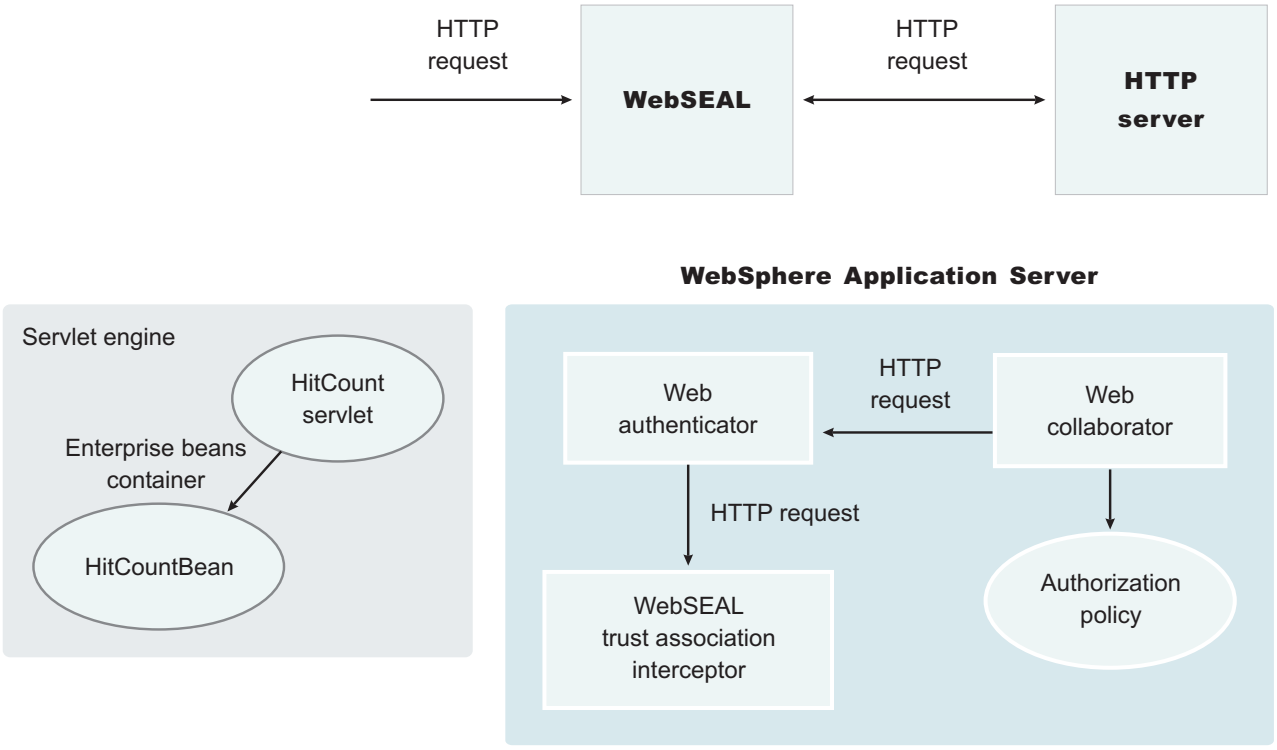
The policy director delegates all of the Web requests to its Web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server consults a Lightweight Directory Access Protocol (LDAP) directory. It can also map the original user ID to another user ID, such as when global single sign-on (GSO) is used.



For successful authentication, the server plays the role of a client to WebSphere Application Server when channeling the request. The server needs its own user ID and password to identify itself to WebSphere Application Server. This identity must be valid in the security realm of WebSphere Application Server. The WebSEAL server replaces the basic authentication information in the HTTP request with its own user ID and password. In addition, WebSphere Application Server must determine the credentials of the requesting client so that the application server has an identity to use as a basis for its authorization decisions. This information is transmitted through the HTTP request by creating a header called `iv-creds`, with the Tivoli Access Manager user credentials as its value.

HTTP server

The junction that is created in the WebSEAL server must get to the HTTP server that serves as the product front end. However, the HTTP server is shielded from knowing that trust association is used. As far as it is concerned, the WebSEAL product is just another HTTP client, and as part of its normal routines, it sends the HTTP request to the product. The only requirement on the HTTP server is a Secure Sockets Layer (SSL) configuration using server authentication only. This requirement protects the requests that flow within the junction.



Web collaborator

When trust association is enabled, the Web collaborator manages the interceptors that are configured in the system. The Web collaborator loads and initializes these interceptors when you restart your servers. When a request is passed to WebSphere Application Server by the Web server, the Web collaborator eventually receives the request for a security check. Two actions must take place:

1. The request must be authenticated.
2. The request must be authorized.

The Web authenticator is called to authenticate the request by passing the HTTP request. If successful, a good credential record is returned by the authenticator, which the Web collaborator uses to base its authorization for the requested resource. If the authorization succeeds, the Web collaborator indicates to WebSphere Application Server that the security check has succeeded and that the requested resource can be served.

Web authenticator

The Web authenticator is asked by the Web collaborator to authenticate a given HTTP request. Knowing that trust association is enabled, the task of the Web authenticator is to find the appropriate trust association interceptor to direct the request for processing. The Web authenticator queries every available interceptor. If no target interceptor is found, the Web authenticator processes the request as though trust association is not enabled.

For an HTTP request sent by the WebSEAL server, the WebSEAL trust association interceptor replies with a positive response to the Web authenticator. Subsequently, the interceptor is asked to validate its trust association with the WebSEAL server and retrieve the Subject, using the new trust association interceptor (TAI) interface, or user ID, using the old TAI interface, of the original user client.

Note: The new Trust Association Interceptor (TAI) interface, com.ibm.wsspi.security.tai.TrustAssociationInterceptor, supports several new features and is different from the existing com.ibm.websphere.security.TrustAssociationInterceptor interface.

WebSphere Application Server Version 4 through WebSphere Application Server Version 5.*x* support the com.ibm.websphere.security.TrustAssociationInterceptor.java interface. WebSphere Application Server Version 6.0.*x* and later supports the com.ibm.wsspi.security.tai.TrustAssociationInterceptor interface.

Trust association interceptor interface

The intent of the trust association interceptor interface is to have reverse proxy security servers (RPSS) exist as the exposed entry points to perform authentication and coarse-grained authorization, while WebSphere Application Server enforces further fine-grained access control. Trust associations improve security by reducing the scope and risk of exposure.

In a typical e-business infrastructure, the distributed environment of a company consists of Web application servers, Web servers, existing systems, and one or more RPSS, such as the Tivoli WebSEAL product. Such reverse proxy servers, front-end security servers, or security plug-ins registered within Web servers, guard the HTTP access requests to the Web servers and the Web application servers. While protecting access to the Uniform Resource Identifiers (URIs), these RPSS perform authentication, coarse-grained authorization, and request routing to the target application server.

Using the trust association interceptor feature

The following points further describe the benefits of the trust association interceptor (TAI) feature:

- RPSS can authenticate WebSphere Application Server users up front and send credential information about the authenticated user to the product so that the product can trust the RPSS to perform authentication and not prompt the end user for authentication data later. The strength of the trust relationship between RPSS and the product is based on the criteria of trust association that is particular to RPSS and enforced through the TAI implementation. This level of trust might need relaxing based on the environment. Be aware of the vulnerabilities in cases where the RPSS is not trusted, based on a security technology.
- The end user credentials most likely are sent in a special format as part of the Hypertext Transfer Protocol (HTTP) headers as in the case of RPSS authentication. The credentials can be a special header or a cookie. The data that passes is implementation specific, and the TAI feature considers this fact and accommodates the idea. The TAI implementation works with the credential data and returns a Subject, using the new TAI interface, or a user ID, using the old TAI interface that represents the end user. WebSphere Application Server uses the information to enforce security policies.

Single sign-on

With single sign-on (SSO) support, Web users can authenticate once when accessing both WebSphere Application Server resources, such as HTML, JavaServer Pages (JSP) files, servlets, enterprise beans, and Lotus Domino resources, such as documents in a Domino database, or accessing resources in multiple WebSphere Application Server domains.

Application servers distributed in multiple nodes and cells can securely communicate using the Lightweight Third Party Authentication (LTPA) protocol. LTPA is intended for distributed, multiple application server and machine environments. LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

LTPA also provides the SSO feature wherein a user is required to authenticate only once in a domain name system (DNS) domain and can access resources in other WebSphere Application Server cells without getting prompted. Web users can authenticate once to a WebSphere Application Server or to a Domino server. This authentication is accomplished by configuring WebSphere Application Servers and the Domino servers to share authentication information.

Without logging in again, Web users can access other WebSphere Application Servers or Domino servers in the same DNS domain that are enabled for SSO. You can enable SSO among WebSphere Application Servers by configuring SSO for WebSphere Application Server. To enable SSO between WebSphere Application Servers and Domino servers, you must configure SSO for both WebSphere Application Server and for Domino.

Prerequisites and conditions

To take advantage of support for SSO between WebSphere Application Servers or between WebSphere Application Server and a Domino server, applications must meet the following prerequisites and conditions:

- Verify that all servers are configured as part of the same DNS domain. The realm names on each system in the DNS domain are case sensitive and must match identically. For example, if the DNS domain is specified as `mycompany.com`, then SSO is effective with any Domino server or WebSphere Application Server on a host that is part of the `mycompany.com` domain, for example, `a.mycompany.com` and `b.mycompany.com`.
- Verify that all servers share the same registry.

  This registry can be either a supported Lightweight Directory Access Protocol (LDAP) directory server or, if SSO is configured between two WebSphere Application Servers, a standalone custom registry.Domino servers do not support standalone custom registries, but you can use a Domino-supported registry as a standalone custom registry within WebSphere Application Server.

  You can use a Domino directory that is configured for LDAP access or other LDAP directories for the registry. The LDAP directory product must have WebSphere Application Server support. Supported products include both Domino and LDAP servers, such as IBM Tivoli Directory Server. Regardless of the choice to use an LDAP or a standalone custom registry, the SSO configuration is the same. The difference is in the configuration of the registry.
- Define all users in a single LDAP directory. Using LDAP referrals to connect more than one directory together is not supported. Using multiple Domino directory assistance documents to access multiple directories also is not supported.
- Enable HTTP cookies in browsers because the authentication information that is generated by the server is transported to the browser in a cookie. The cookie is used to propagate the authentication information for the user to other servers, exempting the user from entering the authentication information for every request to a different server.
- For a Domino server:
  - Domino Release 6.5.4 for iSeries and other platforms are supported.
  - A Lotus Notes client Release 5.0.5 or later is required for configuring the Domino server for SSO.
  - You can share authentication information across multiple Domino domains.
- For WebSphere Application Server:
  - WebSphere Application Server Version 3.5 or later for all platforms are supported.
  - You can use any HTTP Web server that is supported by WebSphere Application Server.
  - You can share authentication information across multiple product administrative domains.
  - Basic authentication (user ID and password) using the basic and form-login mechanisms is supported.
  - By default, WebSphere Application Server does a case-sensitive comparison for authorization. This comparison implies that a user who is authenticated by Domino matches the entry exactly (including the base distinguished name) in the WebSphere Application Server authorization table. If case sensitivity is not considered for the authorization, enable the **Ignore Case** property in the LDAP user registry settings.

Single sign-on for HTTP requests using SPNEGO:

WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server.

SPNEGO is a standard specification defined in The Simple and Protected GSS-API Negotiation Mechanism (IETF RFC 2478).

When WebSphere Application Server administrative security is enabled, the SPNEGO TAI is initialized. While processing inbound HTTP requests, the Web authenticator component interacts with the SPNEGO TAI, which is defined and enabled in the security configuration repository. One interceptor is selected and is responsible for authenticating access to the secured resource that is identified in the HTTP request.

Important: The use of TAIs is an optional feature. If no TAI is selected, the authentication process continues normally.

HTTP users log in and authenticate only once at their desktop and are subsequently authenticated (internally) with WebSphere Application Server. The SPNEGO TAI is invisible to the end-user of WebSphere applications. The SPNEGO TAI is only visible to the Web administrator who is responsible for ensuring a proper configuration, capacity, and maintenance of the Web environment.

In addition to WebSphere Application Server security runtime services, some external components are required to completely enable operation of the SPNEGO TAI. The external components include:

- Microsoft's Windows 2000 or Windows 2003 Servers with Active Directory domain and associated Kerberos Key Distribution Center (KDC).
- A client application, for example, a browser or .NET client, that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478. Microsoft Internet Explorer Version 5.5 or later and Mozilla Firefox Version 1.0 are browser examples. Any browser needs to be configured to use the SPNEGO mechanism. For more information on performing this configuration, see "Configuring SPNEGO TAI in WebSphere Application Server" on page 1253

The authentication of HTTP requests is triggered by the requestor (the client-side), which generates a SPNEGO token. WebSphere Application Server receives this token and validates trust between the requester and WebSphere Application Server. Specifically, the SPNEGO TAI decodes and retrieves the requester's identity from the SPNEGO token. The identity is used to establish a secure context between the requester and the application server.

Remember: The SPNEGO TAI is a server-side solution in WebSphere Application Server. Client-side applications are responsible for generating the SPNEGO token for use by the SPNEGO TAI. The requester's identity in WebSphere Application Server security registry must be identical to that identity the SPNEGO TAI retrieves. An identical match does occur when Microsoft Windows Active Directory server is the Lightweight Directory Access Protocol (LDAP) server that is used in WebSphere Application Server. A custom login module is available as a plug-in to support custom mapping of the identity from the Active Directory to the WebSphere Application Server security registry. See "Mapping user Ids from client to server for SPNEGO" on page 1267 for details on using this custom login module.

WebSphere Application Server validates the identity against its security registry and, if the validation is successful, produces a Lightweight Third Party Authentication (LTPA) security token and places and returns a cookie to the requester in the HTTP response. Subsequent HTTP requests from this same requester to access additional secured resources in WebSphere Application Server use the LTPA security token previously created, to avoid repeated login challenges.

The challenge-response handshake process is illustrated in the following graphic:

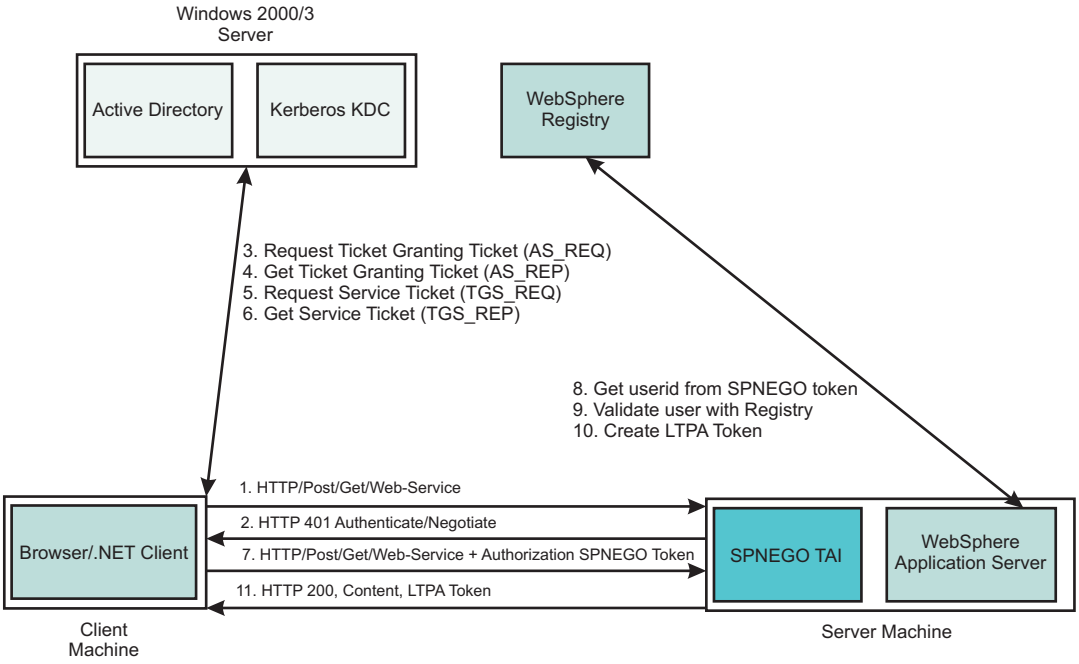


Figure 12. HTTP request processing, WebSphere Application Server - SPNEGO TAI

The SPNEGO TAI can be enabled for all or for selected WebSphere Application Servers in a WebSphere Application Server cell configuration. Also, the behavior of each SPNEGO TAI instance is controlled by custom configuration properties that are used to identify, for example, the criteria used to filter HTTP requests, such as the host name and security realm name used to construct the Kerberos Service Principal Name (SPN). For more information regarding establishing and setting the SPNEGO TAI custom configuration properties, see the following topics:

- Setting up the Kerberos configuration properties. See “Kerberos configuration requirements for SPNEGO TAI” on page 1204.
- Setting or adjusting the SPNEGO TAI custom attributes. See SPNEGO TAI custom configuration attributes.
- Adjusting the SPNEGO TAI filter settings. See Filtering HTTP requests for SPNEGO TAI
- Using the custom login module to map the identity from the Active Directory to the WebSphere Application Server registry. See Mapping user Ids from client to server for SPNEGO.
- Setting the major and additional Java virtual machine (JVM) attributes. See SPNEGO TAI JVM configuration attributes

The Web administrator has access to the following SPNEGO TAI security components and associated configuration data, as illustrated in the following graphic.

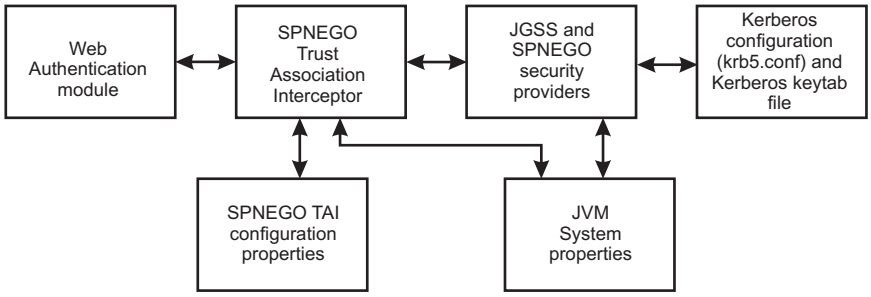


Figure 13. SPNEGO TAI security and configuration elements

- The Web authentication module and the Lightweight Third Party Authentication (LTPA) mechanism provide the plug-in runtime framework for trust association interceptors. See “Configuring the Lightweight Third Party Authentication mechanism” on page 1235 for more detail is configuring the LTPA mechanism for use with the SPNEGO TAI.
- The Java Generic Security Service (JGSS) provider is included in the Java SDK (`jre/lib/ibmjgssprovider.jar`) and used to obtain the Kerberos security context and credentials that are used for authentication. IBM JGSS 1.0 is a Java Generic Security Service Application Programming Interface (GSSAPI) framework with Kerberos V5 as the underlying default security mechanism. GSSAPI is a standardized abstract interface under which can be plugged different security mechanisms based on private-key, public-key and other security technologies. GSSAPI shields secure applications from the complexities and peculiarities of the different underlying security mechanisms. GSSAPI provides identity and message origin authentication, message integrity, and message confidentiality. For more information, see JGSS
- The Kerberos configuration properties (`krb5.conf` or `krb5.ini`) and Kerberos encryption keys (stored in a Kerberos keytab file) are used to establish secure mutual authentication.

  The Kerberos key table manager (Ktab), which is part of JGSS, allows you to manage the principal names and service keys stored in a local Kerberos keytab file. Principal name and key pairs listed in the Kerberos keytab file allow services running on a host to authenticate themselves to the Kerberos Key Distribution Center (KDC). Before a server can use Kerberos, a Kerberos keytab file must be initialized on the host that runs the server.

  “Kerberos configuration requirements for SPNEGO TAI” on page 1204 highlights the Kerberos configuration requirements for the SPNEGO TAI as well as the use of Ktab.
- The SPNEGO provider supplies the implementation of the SPNEGO authentication mechanism, located at `/$WAS_HOME/java/jre/lib/ext/ibmspnego.jar`.
- The custom configuration properties control the runtime behavior of the SPNEGO TAI. Configuration operations are performed with the administrative console or scripting facilities. Refer to “SPNEGO TAI custom configuration attributes” on page 1254 for more information about these custom configuration properties.
- Java virtual machine (JVM) system properties control diagnostic trace information for problem determination of the JGSS security provider and use of the property reload feature.“SPNEGO TAI JVM configuration attributes” on page 1264 describes these JVM configuration attributes

The benefits of having WebSphere Application Server use the SPNEGO TAI include:

- An integrated single sign-on environment with Microsoft Windows 2000 or 2003 Servers using Active Directory domain is established.
- The cost of administering a large number of ids and passwords is reduced.
- A secure and mutually authenticated transmission of security credentials from the Web browser or .NET clients is established.
- Interoperability with Web services and .NET applications that use SPNEGO authentication at the transport level is achieved.

Using the SPNEGO TAI in your WebSphere Application Server environment requires planning then implementation. See “Configuring single sign-on capability with SPNEGO TAI” on page 1248 for the steps to take in planning for SPNEGO TAI. Implementing the use of the SPNEGO TAI is divided into the following areas responsibility:

End user
The end user must configure the Web browser or .NET application to issue HTTP requests that are processed by the SPNEGO TAI.

Web administrator
The Web administrator is responsible for configuring the SPNEGO TAI of WebSphere Application Server to respond to HTTP requests of the client.

WebSphere Application Server administrator
The WebSphere Application Server administrator is responsible for configuring WebSphere Application Server and the SPNEGO TAI for optimum installation performance.

See “Configuring WebSphere Application Server environment to use SPNEGO” on page 1249 for an explanation of the tasks required to use the SPNEGO TAI and the responsible party associated with each task.

SPNEGO TAI configuration requirements:

The configuration that is used by the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) on each selected application server is governed by various system requirements.

The following list of configuration requirements highlights those attributes, properties, qualities, restrictions, exclusions, inclusions, and dependencies that you need to be aware of when planning a WebSphere Application Server configuration that incorporates the use of the SPNEGO TAI.

Table 10. SPNEGO TAI requirements

Function item	Description
SPNEGO TAI	The SPNEGO TAI is a server side solution in WebSphere Application Server. Client-side applications are responsible for generating the SPNEGO token for use by the SPNEGO TAI.
Microsoft Windows	Windows 2000 or Windows 2003 servers with Active Directory domain and its associated Kerberos key distribution center (KDC) is required.
Client application (browser or .NET client)	A browser (client application) or .NET client that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478 is required.
Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)	SPNEGO authentication, as defined in IETF RFC 2478 is used.
Internet browsers	• Use Microsoft Internet Explorer version 5.5 or higher • Use Mozilla Firefox version 1.0
Kerberos Level	Kerberos version 5 is required.
WebSphere Application Server	Version 6.1 is required.
Java generic security service (JGSS)	Version 1.0.1 is required.
Java Virtual Machine (JVM)	See Configuring the JVM for information on configuring JVM.
Java SDK level	Java 5.0 SDK is required.
Encryption Types	RC4-HMAC encryption is only supported when using a Windows 2003 Server as Kerberos key distribution center (KDC) and is not supported with a Windows 2000 Server.

Kerberos configuration requirements for SPNEGO TAI:

Kerberos configuration settings, the Kerberos key distribution center (KDC) name, and realm settings for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) are provided in the Kerberos configuration file or through `java.security.krb5.kdc` and `java.security.krb5.realm` system property files.

The Web administrator creates the Kerberos configuration file with the appropriate settings that allow HTTP requests to be processed by the SPNEGO TAI. See “Creating the Kerberos configuration file for use with the SPNEGO TAI” on page 1251 for more information.

The Web administrator can provide the same Kerberos configuration system properties in separate files: `java.security.krb5.kdc and java.security.krb5.realm.`. See Kerberos Requirements for information on how this is accomplished.

The Kerberos key table manager command (Ktab) allows the Web administrator to manage the principal names and service keys stored in a local Kerberos keytab file. Kerberos service principal (SPN) name and keys listed in the Kerberos keytab file allow services running on the host to authenticate themselves to the KDC. Before SPNEGO TAI can use Kerberos, the WebSphere Application Server administrator must setup a Kerberos keytab file on the host running WebSphere Application Server.

Important: It is very important to protect the keytab files, making them readable only by the authorized WebSphere users.

Important: Any updates to the Kerberos keytab file using Ktab do not affect the Kerberos database. If you change the keys in the Kerberos keytab file, you must also make the corresponding changes to the Kerberos database.

Below is an example of how Ktab is used on a LINUX platform to add new principal names to the Kerberos keytab file.

```
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab -a
HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM ot56prod -k /etc/krb5.keytab
Done!
Service key for principal HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM saved
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab
1 entries in keytab, name: /etc/krb5.keytab
        KVNO    Principal
        ----    ---------

        1       HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
[root@wssecjibe bin]# ls /etc/krb5.*
/etc/krb5.conf       /etc/krb5.ini.orig  /etc/krb5.keytab.good
/etc/krb5.conf.orig  /etc/krb5.keytab
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab -a
HTTP/wssecredhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM ot56prod -k /etc/krb5.keytab
Done!
Service key for principal HTTP/wssecredhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
saved
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab
2 entries in keytab, name: /etc/krb5.keytab
        KVNO    Principal
        ----    ---------

        1       HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM

        1       HTTP/wssecredhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
```

Tip: On WebSphere Application Server, Ktab is located at:

```
<install root>/java/jre/bin
```

Global single sign-on principal mapping:

You can use the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager to manage authentication to enterprise information systems (EIS) such as databases, transaction processing systems, and message queue systems that are located within the WebSphere Application Server security domain. Such authentication is achieved using the global single sign-on (GSO) principal mapper Java Authentication and Authorization Service (JAAS) login module for Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture resources.

With GSO principal mapping, a special-purpose JAAS login module inserts a credential into the subject header. This credential is used by the resource adapter to authenticate to the EIS. The JAAS login module used is configured on a per-connection factory basis. The default principal mapping module retrieves the user name and password information from XML configuration files. The JACC provider for Tivoli Access Manager bypasses the credential that is stored in the Extensible Markup Language (XML) configuration files and uses the Tivoli Access Manager global sign-on (GSO) database instead to provide the authentication information for the EIS security domain.

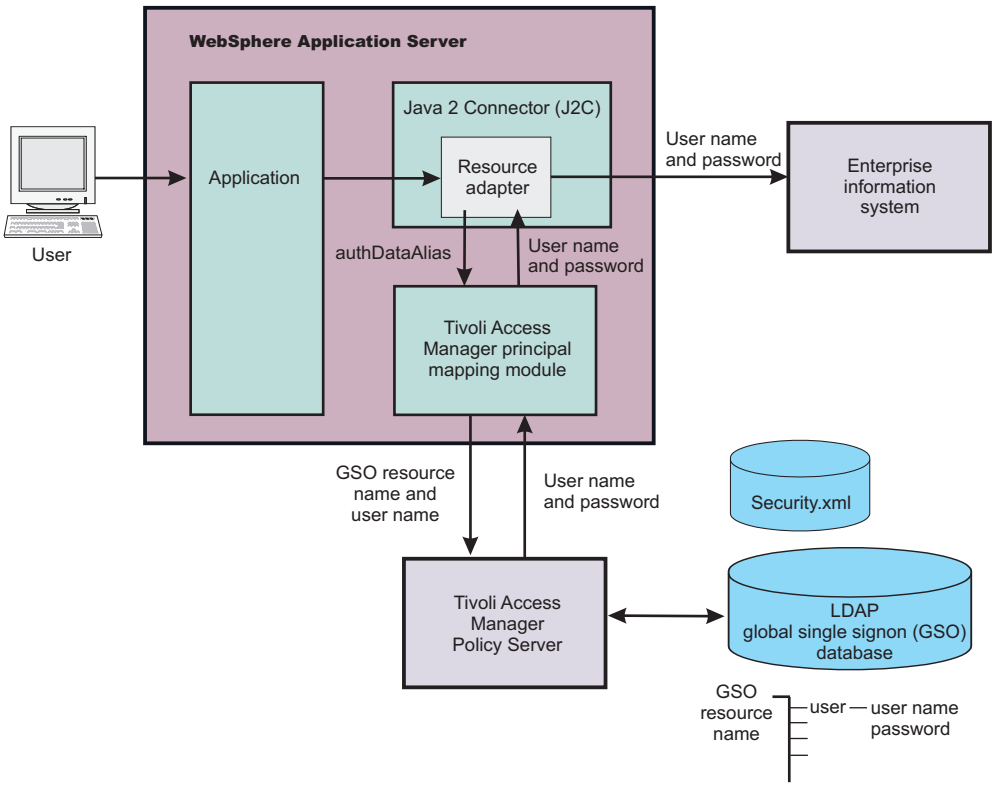
WebSphere Application Server provides a default principal mapping module that associates user credential information with EIS resources. The default mapping module is defined in the WebSphere Application Server administrative console on the Application login panel. To access the panel, click **Security > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **Application logins**. The mapping module name is DefaultPrincipalMapping.

The EIS security domain user ID and password are defined under each connection factory by an authDataAlias attribute. The authDataAlias attribute does not contain the user name and password; this attribute contains an alias that refers to a user name and password pair that is defined elsewhere.

The Tivoli Access Manager principal mapping module uses the authDataAlias attribute to determine the GSO resource name and the user name that is required to perform the lookup on the Tivoli Access Manager GSO database. The Tivoli Access Manager Policy Server retrieves the GSO data from the user registry.

Tivoli Access Manager stores authentication information on the Tivoli Access Manager GSO database against a resource and user name pair.

GSO principal mapping architecture



Security attribute propagation

With *Security attribute propagation*, WebSphere Application Server can transport security attributes (authenticated Subject contents and security context information) from one server to another in your configuration. WebSphere Application Server might obtain these security attributes from either an enterprise user registry, which queries static attributes, or a custom login module, which can query static or dynamic attributes. Dynamic security attributes, which are custom in nature, might include the authentication strength that is used for the connection, the identity of the original caller, the location of the original caller, the IP address of the original caller, and so on.

Security attribute propagation provides propagation services using Java serialization for any objects that are contained in the Subject. However, Java code must be able to serialize and deserialize these objects. The Java programming language specifies the rules for how Java code can serialize an object. Because problems can occur when dealing with different platforms and versions of software, WebSphere Application Server also offers a token framework that enables custom serialization functionality. The token framework has other benefits that include the ability to identify the uniqueness of the token. This uniqueness determines how the Subject gets cached and the purpose of the token. The token framework defines four marker token interfaces that enable the WebSphere Application Server runtime to determine how to propagate the token.

Important: Any custom tokens that are used in this framework are not used by WebSphere Application Server for authorization or authentication. The framework serves as a way to notify WebSphere Application Server that you want these tokens propagated in a particular way. WebSphere Application Server handles the propagation details, but does not handle serialization or deserialization of custom tokens. Serialization and deserialization of these custom tokens are carried out by the implementation and handled by a custom login module.

With WebSphere Application Server Version 6.0 and later, a custom Java Authorization Contract for Container (JACC) provider can be configured to enforce access control for Java 2

Platform, Enterprise Edition (J2EE) applications. A custom JACC provider can explore the custom security attributes in the caller JAAS subject in making access control decisions.

When a request is being authenticated, a determination is made by the login modules whether this request is an *initial login* or a *propagation login*. An initial login is the process of authenticating the user information, typically a user ID and password, and then calling the application programming interfaces (APIs) for the remote user registry to look up secure attributes that represent the user access rights. A propagation login is the process of validating the user information, typically a Lightweight Third Party Authentication (LTPA) token, and then deserializing a series of tokens that constitute both custom objects and token framework objects known to WebSphere Application Server.

The following marker tokens are introduced in the framework:

Authorization token

The authorization token contains most of the authorization-related security attributes that are propagated. The default authorization token is used by the WebSphere Application Server authorization engine to make Java 2 Platform, Enterprise Edition (J2EE) authorization decisions. Service providers can use custom authorization token implementations to isolate their data in a different token, perform custom serialization and de-serialization, and make custom authorization decisions using the information in their token at the appropriate time. For information on how to use and implement this token type, see “Default propagation token” on page 1210 and Implementing a custom propagation token.

Single sign-on (SSO) token

A custom SingleSignonToken token that is added to the Subject is automatically added to the response as an HTTP cookie and contains the attributes sent back to Web browsers. The token interface getName method with the getVersion method defines the cookie name. WebSphere Application Server defines a default SingleSignonToken token with the LtpaToken name and Version 2. The cookie name added is LtpaToken2. Do not add sensitive information, confidential information, or unencrypted data to the response cookie.

It is also recommended that any time that you use cookies, use the Secure Sockets Layer (SSL) protocol to protect the request. Using an SSO token, Web users can authenticate once when accessing Web resources across multiple WebSphere Application Servers. A custom SSO token extends this functionality by adding custom processing to the single sign-on scenario. For more information on SSO tokens, see “Implementing single sign-on to minimize Web user authentications” on page 1245. For information on how to use and implement this token type, see “Default single sign-on token” on page 1219 and Implementing a custom single sign-on token.

Propagation token

The propagation token is not associated with the authenticated user so it is not stored in the Subject. Instead, the propagation token is stored on the thread and follows the invocation wherever it goes. When a request is sent outbound to another server, the propagation tokens on that thread are sent with the request and the tokens are run by the target server. The attributes that are stored on the thread are propagated regardless of the Java 2 Platform, Enterprise Edition (J2EE) RunAs user switches.

The default propagation token monitors and logs all user switches and host switches. You can add additional information to the default propagation token using the WSSecurityHelper application programming interfaces (APIs). To retrieve and set custom implementations of a propagation token, you can use the WSSecurityPropagationHelper class. For information on how to use and implement this token type, see “Default propagation token” on page 1210 and Implementing a custom propagation token.

Authentication token

The authentication token flows to downstream servers and contains the identity of the user. This token type serves the same function as the Lightweight Third Party Authentication (LTPA) token in

previous versions. Although this token type is typically reserved for internal WebSphere Application Server purposes, you can add this token to the Subject and the token is propagated using the getBytes method of the token interface.

A custom authentication token is used solely for the purpose of the service provider that adds it to the Subject. WebSphere Application Server does not use it for authentication purposes because a default authentication token exists that is used for WebSphere Application Server authentication. This token type is available for the service provider to identify how the custom data uses the token to perform custom authentication decisions. For information on how to use and implement this token type, see “Default authentication token” on page 1220 and Implementing a custom authentication token.

Horizontal propagation versus downstream propagation

In WebSphere Application Server, both horizontal propagation, which uses single sign-on for Web requests, and downstream propagation, which uses Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) to access enterprise beans, are available.

Horizontal propagation

In horizontal propagation, security attributes are propagated among front-end servers. The serialized security attributes, which are the Subject contents and the propagation tokens, can contain both static and dynamic attributes. The single sign-on (SSO) token stores additional system-specific information that is needed for horizontal propagation. The information contained in the SSO token tells the receiving server where the originating server is located and how to communicate with that server. Additionally, the SSO token also contains the key to look up the serialized attributes. To enable horizontal propagation, you must configure the single sign-on token and the Web inbound security attribute propagation features. You can configure both of these features using the administrative console.

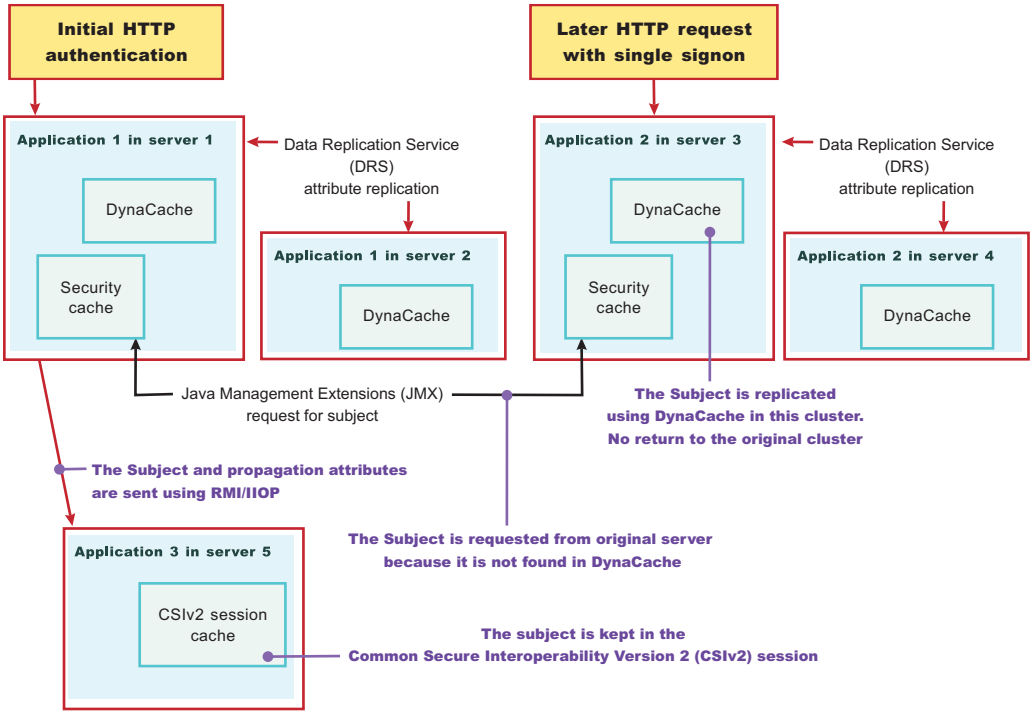
When front-end servers are configured and in the same distributed replication service (DRS) replication domain, the application server automatically propagates the serialized information to all of the servers within the same domain. In figure 1, application 1 is deployed on server 1 and server 2, and both servers are members of the same DRS replication domain. If a request originates from application 1 on server 1 and then gets redirected to application 1 on server 2, the original login attributes are found on server 2 without additional remote requests.

However, if the request originates from application 1 on either server 1 or server 2, but the request is redirected to application 2 on either server 1 or server 2, the serialized information is not found in the DRS cache because the servers are not configured in the same replication domain. As a result, a remote Java Management Extensions (JMX) request is sent back to the originating server that hosts application 1 to obtain the serialized information so that original login information is available to the application. By getting the serialized information using a single JMX remote call back to the originating server, the following benefits are realized:

- You gain the function of retrieving login information from the original server.
- You do not need to perform any remote user registry calls because the application server can regenerate the Subject from the serialized information. Without this ability, the application server might make five to six separate remote calls.

Figure 1

1. User authenticates to server 1.
2. Server 1 makes an RMI request to server 5.
3. User accesses another Web application on server 3.



Performance implications for horizontal propagation

The performance implications of either the DRS or JMX remote call depends upon your environment. THE DRS or JMX remote call is used for obtaining the original login attributes. Horizontal propagation reduces many of the remote user registry calls in cases where these calls cause the most performance problems for an application. However, the de-serialization of these objects also might cause performance degradation, but this degradation might be less than the remote user registry calls. It is recommended that you test your environment with horizontal propagation enabled and disabled. In cases where you must use horizontal propagation for preserving original login attributes, test whether DRS or JMX provides better performance in your environment. Typically, it is recommended that you configure DRS both for failover and performance reasons. However, because DRS propagates the information to all of the servers in the same replication domain (whether the servers are accessed or not), there might be a performance degradation if too many servers are in the same replication domain. In this case, either reduce the number of servers in the replication domain or do not configure the servers in a DRS replication domain. The later suggestion causes a JMX remote call to retrieve the attributes, when needed, which might be quicker overall.

Downstream propagation

In *downstream propagation*, a Subject is generated at the Web front-end server, either by a propagation login or a user registry login. WebSphere Application Server propagates the security information downstream for enterprise bean invocations when both Remote Method Invocation (RMI) outbound and inbound propagation are enabled.

Benefits of propagating security attributes

The security attribute propagation feature of WebSphere Application Server has the following benefits:

- Enables WebSphere Application Server to use the security attribute information for authentication and authorization purposes. The propagation of security attributes can eliminate the need for user registry

calls at each remote hop along an invocation. Previous versions of WebSphere Application Server propagated only the user name of the authenticated user, but ignored other security attribute information that needed to be regenerated downstream using remote user registry calls. To accentuate the benefits of this new functionality, consider the following example:

In previous releases, you might use a reverse proxy server (RPSS), such as WebSEAL, to authenticate the user, gather group information, and gather other security attributes. As stated previously, WebSphere Application Server accepted the identity of the authenticated user, but disregarded the additional security attribute information. To create a Java Authentication and Authorization Service (JAAS) Subject containing the needed WSCredential and WSPrincipal objects, WebSphere Application Server made 5 to 6 calls to the user registry. The WSCredential object contains various security information that is required to authorize a J2EE resource. The WSPrincipal object contains the realm name and the user that represents the principal for the Subject.

In the current release of the Application Server, information that is obtained from the reverse proxy server can be used by WebSphere Application Server and propagated downstream to other server resources without additional calls to the user registry. The retaining of the security attribute information enables you to protect server resources properly by making appropriate authorization and trust-based decisions User switches that occur because of J2EE RunAs configurations do not cause the application server to lose the original caller information. This information is stored in the PropagationToken located on the running thread.

- Enables third-party providers to plug in custom tokens. The token interface contains a getBytes method that enables the token implementation to define custom serialization, encryption methods, or both.
- Provides the ability to have multiple tokens of the same type within a Subject created by different providers. WebSphere Application Server can handle multiple tokens for the same purpose. For example, you might have multiple authorization tokens in the Subject and each token might have distinct authorization attributes that are generated by different providers.
- Provides the ability to have a unique ID for each token type that is used to formulate a more unique subject identifier than just the user name in cases where dynamic attributes might change the context of a user login. The token type has a getUniqueId() method that is used for returning a unique string for caching purposes. For example, you might need to propagate a location ID, which indicates the location from which the user logs into the system. This location ID can be generated during the original login using either an reverse proxy server or the WEB_INBOUND login configuration and added to the Subject prior to serialization. Other attributes might be added to the Subject as well and use a unique ID. All of the unique IDs must be considered for the uniqueness of the entire Subject. WebSphere Application Server has the ability to specify what is unique about the information in the Subject, which might affect how the user accesses the Subject later.

Default propagation token:

A default propagation token is located on the running thread for applications and the security infrastructure to use. WebSphere Application Server propagates this default propagation token downstream and the token stays on the thread where the invocation lands at each hop.

The data is available from within the container of any resource where the propagation token lands. Remember that you must enable the propagation feature at each server where a request is sent for propagation to work. Make sure that you enable security attribute propagation for all of the cells in your environment where you want propagation

There is a WSSecurityHelper class that has application programming interfaces (APIs) for accessing the PropagationToken attributes. This topic documents the usage scenarios and includes examples. A close relationship exists between the propagation token and the work area feature. The main difference between these features is that after you add attributes to the propagation token, you cannot change the attributes. You cannot change these attributes so that the security runtime can add auditable information and have that information remain there for the life of the invocation. Any time that you add an attribute to a specific key, an ArrayList object is stored to hold that attribute. Any new attribute that is added with the same key

is added to the ArrayList object. When you call getAttributes, the ArrayList object is converted to a String array and the order is preserved. The first element in the String array is the first attribute added for that specific key.

In the default propagation token, a change flag is kept that logs any data changes to the token. These changes are tracked to enable WebSphere Application Server to know when to send the authentication information downstream again so that the downstream server has those changes. Normally, Common Secure Interoperability Version 2 (CSIv2) maintains a session between servers for an authenticated client. If the propagation token changes, a new session is generated and subsequently a new authentication occurs. Frequent changes to the propagation token during a method cause frequent downstream calls. If you change the token prior to making many downstream calls or you change the token between each downstream call, you might impact security performance.

Getting the server list from the default propagation token

Every time the propagation token is propagated and used to create the authenticated Subject, either horizontally or downstream, the name of the receiving application server is logged into the propagation token. The format of the host is ″Cell:Node:Server″, which provides you access to the cell name, node name, and server name of each application server that receives the invocation. The following code provides you with this list of names and can be called from a Java 2 Platform, Enterprise Edition (J2EE) application:

```
String[] server_list = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
  try
  {
    // Gets the server_list string array
    server_list = com.ibm.websphere.security.WSSecurityHelper.getServerList();
  }
  catch (Exception e)
  {
    // Performs normal exception handling for your application
  }

  if (server_list != null)
  {
    // print out each server in the list, server_list[0] is the first server
    for (int i=0; i<server_list.length; i++)
    {
      System.out.println(″Server[″ + i + ″] = ″ + server_list[i]);
    }
  }
}
```

The format of each server in the list is: *cell*:*node_name*:*server_name*. The output, for example, is: myManager:node1:server1

Getting the caller list from the default propagation token

A default propagation token is generated any time an authenticated user is set on the running thread or anyone tries to add attributes to the propagation token. Whenever an authenticated user is set on the thread, the user is logged in the default propagation token. At times, the same user might be logged in

multiple times if the RunAs user is different from the caller. The following list provides the rules that are used to determine if a user that is added to the thread gets logged into the propagation token:

- The current Subject must be authenticated. For example, an unauthenticated Subject is not logged.
- The current authenticated Subject is logged if a Subject is not previously logged.
- The current authenticated Subject is logged if the last authenticated Subject that is logged does not contain the same user.
- The current authenticated Subject is logged on each unique application server that is involved in the propagation process.

The following code sample shows how to use the getCallerList API:

```
String[] caller_list = null;

// If security is disabled on this application server, do not check the caller list
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
  try
  {
    // Gets the caller_list string array
    caller_list = com.ibm.websphere.security.WSSecurityHelper.getCallerList();
  }
  catch (Exception e)
  {
    // Performs normal exception handling for your application
  }

  if (caller_list != null)
  {
    // Prints out each caller in the list, caller_list[0] is the first caller
    for (int i=0; i<caller_list.length;i++)
    {
      System.out.println("Caller[" + i + "] = " + caller_list[i]);
    }
  }
}
```

The format of each caller in the list is: *cell*:*node_name*:*server_name*:*realm*:*port_number*/*securityName*. The output, for example, is: myManager:node1:server1:ldap.austin.ibm.com:389/jsmith.

Getting the first caller from the default propagation token

Whenever you want to know which authenticated caller started the request, you can call the getFirstCaller method and the caller list is parsed. However, this method returns the security name of the caller only. If you need to know more than the security name, call the getCallerList method and retrieve the first entry in the String array. This entry provides all the caller information. The following code sample retrieves the security name of the first authenticated caller using the getFirstCaller API:

```
String first_caller = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
  try
  {
    // Gets the first caller
```

```
    first_caller = com.ibm.websphere.security.WSSecurityHelper.getFirstCaller();

    // Prints out the caller name
    System.out.println("First caller: " + first_caller);
   }
   catch (Exception e)
   {
    // Performs normal exception handling for your application
   }
 }
```

The output, for example, is: jsmith.

Getting the first application server name from the default propagation token

Whenever you want to know what the first application server is for this request, call the getFirstServer method directly. The following code sample retrieves the name of the first application server using the getFirstServer API:

```
 String first_server = null;

 // If security is disabled on this application server, do not bother checking
 if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
 {
  try
  {
   // Gets the first server
   first_server = com.ibm.websphere.security.WSSecurityHelper.getFirstServer();

   // Prints out the server name
   System.out.println("First server: " + first_server);
  }
  catch (Exception e)
  {
   // Performs normal exception handling for your application
  }
 }
```

The output, for example, is: myManager:node1:server1.

Adding custom attributes to the default propagation token

You can add custom attributes to the default propagation token for application usage. This token follows the request downstream so that the attributes are available when needed. When you use the default propagation token to add attributes, you must understand the following issues:

- Adding information to the propagation token affects CSIv2 session caching. Add information sparingly between remote requests.
- After you add information with a specific key, the information cannot be removed.
- You can add as many values to a specific key as you need. However, all of the values must be available from a returned String array in the order that they were added.
- The propagation token is available only on servers where propagation and security are enabled.
- The Java 2 Security javax.security.auth.AuthPermission wssecurity.addPropagationAttribute attribute is needed to add attributes to the default propagation token.

- An application cannot use keys that begin with either com.ibm.websphere.security or com.ibm.wsspi.security. These prefixes are reserved for system usage.

The following code sample shows how to use the addPropagationAttribute API:

```
// If security is disabled on this application server,
      // do not check the status of server security
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Specifies the key and values
  String key = ″mykey″;
  String value1 = ″value1″;
  String value2 = ″value2″;

  // Sets key, value1
       com.ibm.websphere.security.WSSecurityHelper.
       addPropagationAttribute (key, value1);

  // Sets key, value2
  String[] previous_values = com.ibm.websphere.security.WSSecurityHelper.
            addPropagationAttribute (key, value2);

  // Note: previous_values should contain value1
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }
}
```

See “Getting custom attributes from the default propagation token” to retrieve attributes using the getPropagationAttributes application programming interface (API).

Getting custom attributes from the default propagation token

Custom attributes are added to the default propagation token using the addPropagationAttribute API. Retrieve these attributes using the getPropagationAttributes API. This token follows the request downstream so the attributes are available when needed. When you use the default propagation token to retrieve attributes, you must understand the following issues:

- The propagation token is available only on servers where propagation and security are enabled.
- The Java 2 Security javax.security.auth.AuthPermission ″wssecurity.getPropagationAttributes″ permission is needed to retrieve attributes from the default propagation token.

The following code sample shows how to use the getPropagationAttributes API:

```
// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  String key = ″mykey″;
  String[] values = null;
```

```
    // Sets key, value1
          values = com.ibm.websphere.security.WSSecurityHelper.
          getPropagationAttributes (key);

    // Prints the values
    for (int i=0; i<values.length; i++)
    {
      System.out.println("Value[" + i + "] = " + values[i]);
    }
  }
  catch (Exception e)
  {
    // Performs normal exception handling for your application
  }
}
```

The output, for example, is:

```
Value[0] = value1
Value[1] = value2
```

See Adding custom attributes to the default PropagationToken to add attributes using the addPropagationAttributes API.

Changing the token factory that is associated with the default propagation token

When WebSphere Application Server generates a default propagation token, the Application Server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.propagationTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The default token factory that is specified for this property is called com.ibm.ws.security.ltpa.AuthzPropTokenFactory. This token factory encodes the data in the propagation token and does not encrypt the data. Because the propagation token typically flows over CSIv2 using Secure Sockets Layer (SSL), encrypting the token is not required. However, if you need additional security for the propagation token, you can associate a different token factory implementation with this property to get encryption. For example, if you choose to associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory with this property, the token is AES encrypted. However, you need to weigh the performance impacts against your security needs. Adding sensitive information to the propagation token is a good reason to change the token factory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default propagation token, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your token factory implementation instantiates and validates your token implementation. You can choose to use the Lightweight Third Party Authentication (LTPA) keys and have them pass into the initialize method of the token factory, or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default propagation token, using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.

2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.propagationTokenFactory property and verify that the value of this property matches your custom token factory implementation.
4. Verify that your implementation classes are put into the *app_server_root*/classes directory so that the WebSphere Application Server class loader can load the classes.
5. Verify that your implementation classes are located in the ${USER_INSTALL_ROOT}/classes directory so that the WebSphere Application Server class loader can load the classes.

Related tasks

"Propagating security attributes among application servers" on page 1283
Use the security attribute propagation feature of WebSphere Application Server to send security attribute information regarding the original login to other servers using a token. This topic will help to configure WebSphere Application Server to propagate security attributes to other servers.

Default authorization token:

This topic explains how WebSphere Application Server uses the default authorization token. Consider using the default authorization token when you are looking for a place to add string attributes that get propagated downstream.

However, make sure that the attributes you add to the authorization token are specific to the user that is associated with the authenticated Subject. If they are not specific to a user, the attributes probably belong in the propagation token, which is also propagated with the request. For more information on the propagation token, see "Default propagation token" on page 1210. To add attributes into the authorization token, you must plug in a custom login module into the various system login modules that are configured. Any login module configuration that has the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule implementation configured can receive propagated information and can generate propagation information that can be sent outbound to another server.

If propagated attributes are not presented to the login configuration during an initial login, a default authorization token is created in the wsMapDefaultInboundLoginModule login module after the login occurs in the ltpaLoginModule login module. You can obtain a reference to the default authorization token from the login method using the sharedState hashmap. You must plug in the custom login module after the wsMapDefaultInboundLoginModule implementation for WebSphere Application Server to see the default authorization token.

For more information on the Java Authentication and Authorization Service (JAAS) programming model, see Security: Resources for learning.

Important: Whenever you plug a custom login module into the WebSphere Application Server login infrastructure, you must ensure that the code is trusted. When you add the login module into the *app_server_root*/classes directory, it has Java 2 Security AllPermissions permissions. It is recommended that you add your login module and other infrastructure classes into a private directory. However, if you use a private directory, modify the $(WAS_INSTALL_ROOT)/properties/server.policy file so that the private directory, Java archive (JAR) file, or both have the permissions that are needed to run the application programming interfaces (API) that are called from the login module. Because the login module might run after the application code on the call stack, you might consider adding a doPrivileged code block so that you do not need to add additional permissions to your applications.

The following sample code shows you how to obtain a reference to the default authorization token from the login method, how to add attributes to the token, and how to read from the existing attributes that are used for authorization.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
          Map sharedState, Map options)
 {
     // (For more information on initialization, see
     // Custom login module development for a system login configuration.)

  // Get a reference to the sharedState map that is passed in during initialization.
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // Custom login module development for a system login configuration.)

  // Look for the default AuthorizationToken in the shared state
  defaultAuthzToken  = (com.ibm.wsspi.security.token.AuthorizationToken)
       sharedState.get
     (com.ibm.wsspi.security.auth.callback.Constants.WSAUTHZTOKEN_KEY);

  // Might not always have one of these generated. It depends on the login
     // configuration setup.
  if (defaultAuthzToken != null)
  {
   try
   {
    // Add a custom attribute
    defaultAuthzToken.addAttribute("key1", "value1");

    // Determine all of the attributes and values that exist in the token.
    java.util.Enumeration listOfAttributes = defaultAuthorizationToken.
              getAttributeNames();

    while (listOfAttributes.hasMoreElements())
    {
     String key = (String) listOfAttributes.nextElement();

     String[] values = (String[]) defaultAuthorizationToken.getAttributes (key);

     for (int i=0;  i<values.length; i++)
     {
      System.out.println ("Key: " + key + ", Value[" + i + "]: "
                  + values[i]);
     }
    }

    // Read the existing uniqueID attribute.
    String[]  uniqueID = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_UNIQUEID);

     // Getthe uniqueID from the String[]
     String unique_id = (uniqueID != null &&
                uniqueID[0] != null) ? uniqueID[0] : "";

    // Read the existing expiration attribute.
    String[]  expiration = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_EXPIRATION);

     // An example of getting a long expiration value from the string array.
```

```
     long expire_time = 0;
     if (expiration != null && expiration[0] != null)
      expire_time = Long.parseLong(expiration[0]);

    // Read the existing display name attribute.
    String[]  securityName = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_SECURITYNAME);

     // Get the display name from the String[]
     String display_name = (securityName != null &&
                securityName[0] != null) ? securityName[0] : "";


    // Read the existing long securityName attribute.
    String[]  longSecurityName = defaultAuthzToken.getAttributes
     (com.ibm.wsspi.security.token.AttributeNameConstants.
             WSCREDENTIAL_LONGSECURITYNAME);

    // Get the long security name from the String[]
    String long_security_name = (longSecurityName != null &&
              longSecurityName[0] != null) ? longSecurityName[0] : "";


    // Read the existing group attribute.
    String[]  groupList = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_GROUPS);

    // Get the groups from the String[]
    ArrayList groups = new ArrayList();
    if (groupList != null)
    {
     for (int i=0; i<groupList.length; i++)
     {
      System.out.println ("group[" + i + "] = " + groupList[i]);
      groups.add(groupList[i]);
     }
    }
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }

 }

 public boolean commit() throws LoginException
 {
  // (For more information on what to do during commit, see
     // Custom login module development for a system login configuration.)

 }

 private java.util.Map _sharedState = null;
 private com.ibm.wsspi.security.token.AuthorizationToken defaultAuthzToken = null;
}
```

Changing the token factory that is associated with the default authorization token

When WebSphere Application Server generates a default authorization token, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.authorizationTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The com.ibm.ws.security.ltpa.AuthzPropTokenFactory token factory is the default. This token factory encodes the data, but does not encrypt the data in the authorization token. Because the authorization token typically flows over Common Secure Interoperability Version 2 (CSIv2) using Secure Sockets Layer (SSL), encrypting the token is not necessary. However, if you need additional security for the authorization token, you can associate a different token factory implementation with this property to get encryption. For example, if you associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory with this property, the token uses Advanced Encryption Standard (AES) encryption. However, you need to weigh the performance impacts against your security needs. Adding sensitive information to the authorization token is one reason to change the token factory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default authorization token, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your token factory implementation instantiates and validates your token implementation. You can use the Lightweight Third Party Authentication (LTPA) keys that are passed into the initialize method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, that is available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default authorization token, using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.authorizationTokenFactory property and verify that the value of this property matches your custom token factory implementation.
4. Verify that your implementation classes are put into the *app_server_root*/classes directory so that the WebSphere Application Server class loader can load the classes.
5. Verify that your implementation classes are put into the ${USER_INSTALL_ROOT}/classes directory so that the WebSphere Application Server class loader can load the classes.

Default single sign-on token:

Do not use the default single sign-on token in service provider code. This default token is used by the WebSphere Application Server run-time code only.

Size limitations exist for this token when it is added as an HTTP cookie. If you need to create an HTTP cookie using this token framework, you can implement a custom single sign-on token. To implement a custom single sign-on token see Implementing a custom single sign-on token for more information.

Changing the token factory that is associated with the default single sign-on token

When the default single sign-on token is generated, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.singleSignonTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The com.ibm.ws.security.ltpa.LTPAToken2Factory token factory is the default that is specified for this property. This token factory creates a single sign-on (SSO) token called LtpaToken2, which WebSphere Application Server uses for propagation. This token factory uses the AES/CBC/PKCS5Padding cipher. If you change this token factory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to version 5.1.1 that use the default token factory. Only servers running WebSphere Application Server Version 5.1.1 or later with propagation enabled are aware of the LtpaToken2 cookie. If all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new token factory this awareness is not a problem.

If you need to perform your own signing and encryption of the default single sign-on token, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your token factory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the Lightweight Third-Party Authentication (LTPA) keys passed into the initialize method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default single sign-on token using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.singleSignonTokenFactory property and verify that the value of this property matches your custom TokenFactory implementation.
4. Verify that your implementation classes are put into the *app_server_root*/classes directory so that the WebSphere Application Server class loader can load the classes.
5. Verify that your implementation classes are located in the ${USER_INSTALL_ROOT}/classes directory so that the WebSphere Application Server class loader can load the classes.

Related tasks

“Propagating security attributes among application servers” on page 1283
Use the security attribute propagation feature of WebSphere Application Server to send security attribute information regarding the original login to other servers using a token. This topic will help to configure WebSphere Application Server to propagate security attributes to other servers.

Implementing a custom single sign-on token
You can create your own single sign-on token implementation. The single sign-on token implementation is set in the login Subject and added to the HTTP response as an HTTP cookie.

Default authentication token:

Do not use the default authentication token in service provider code. This default token is used by the WebSphere Application Server run-time code only and is authentication mechanism specific.

Any modifications to this token by service provider code can potentially cause interoperability problems. If you need to create an authentication token for custom usage, see Implementing a custom authentication token for more information.

Changing the token factory that is associated with the default authentication token

When WebSphere Application Server generates a default authentication token, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.authenticationTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The com.ibm.ws.security.ltpa.LTPATokenFactory token factory is the default for this property. The LTPATokenFactory token factory uses the DESede/ECB/PKCS5Padding cipher. This token factory creates an interoperable Lightweight Third Party Authentication (LTPA) token. If you change this token factory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to Version 5.1.1 and any other servers that do not support the new token factory implementation. However, if all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new token factory, this interoperability is not a problem.

If you associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory with the com.ibm.wsspi.security.token.authenticationTokenFactory property, the token is Advanced Encryption Standard (AES) encrypted. However, you need to weigh the performance against your security needs. You might add additional attributes to the authentication token in the Subject during a login that are available downstream.

If you need to perform your own signing and encryption of the default authentication token, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your token factory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the LTPA keys that are passed into the initialize method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default authentication token using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.authenticationTokenFactory property and verify that the value of this property matches your custom token factory implementation.
4. Verify that your implementation classes are put into the *install_dir*/classes directory so that the WebSphere Application Server class loader can load the classes.

Simple WebSphere authentication mechanism

The Simple WebSphere authentication mechanism (SWAM) is intended for simple, non-distributed, single application server runtime environments.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

The single application server restriction is due to the fact that SWAM does not support *forwardable* credentials. If a servlet or enterprise bean in application server process 1, invokes a remote method on an enterprise bean living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, can cause authorization failures.

Because SWAM is intended for a single application server process, single sign-on (SSO) is not supported.

The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

UserRegistry interface methods

Implementing this interface enables WebSphere Application Server security to use custom registries. This capability extends the `java.rmi` file. With a remote registry, you can complete this process remotely.

Implementation of this interface must provide implementations for:

- initialize(java.util.Properties)
- checkPassword(String,String)
- mapCertificate(X509Certificate[])
- getRealm
- getUsers(String,int)
- getUserDisplayName(String)
- getUniqueUserId(String)
- getUserSecurityName(String)
- isValidUser(String)
- getGroups(String,int)
- getGroupDisplayName(String)
- getUniqueGroupId(String)
- getUniqueGroupIds(String)
- getGroupSecurityName(String)
- isValidGroup(String)
- getGroupsForUser(String)
- getUsersForGroup(String,int)
- createCredential(String)

```
public void initialize(java.util.Properties props)
      throws CustomRegistryException,
             RemoteException;
```

This method is called to initialize the UserRegistry method. All the properties that are defined in the Custom User Registry panel propagate to this method.

For the `FileRegistrySample.java` sample file, the initialize method retrieves the names of the registry files that contain the user and group information.

This method is called during server bringup to initialize the registry. This method is also called when validation is performed by the administrative console, when security is on. This method remains the same as in Version 4.x.

```
public String checkPassword(String userSecurityName, String password)
      throws PasswordCheckFailedException,
             CustomRegistryException,
             RemoteException;
```

The checkPassword method is called to authenticate users when they log in using a name or user ID and a password. This method returns a string which, in most cases, is the user security name. A credential is created for the user for authorization purposes. This user name is also returned for the getCallerPrincipal enterprise bean call and the servlet calls the getUserPrincipal and getRemoteUser methods. See the getUserDisplayName method for more information if you have display names in your registry. In some situations, if you return a user other than the one who is logged in, you must verify that the user is valid in the registry.

For the FileRegistrySample.java sample file, the mapCertificate method gets the distinguished name (DN) from the certificate chain and makes sure it is a valid user in the registry before returning the user. For the sample, the checkPassword method checks the name and password combination in the user registry and, if they match, the method returns the user being authenticated.

This method is called for various scenarios, for example, by the administrative console to validate the user information after the user registry is initialized. This method is also called when you access protected resources in the product for authenticating the user and before proceeding with the authorization. This method is the same as in Version 4.x.

```
public String mapCertificate(X509Certificate[] cert)
      throws CertificateMapNotSupportedException,
             CertificateMapFailedException,
             CustomRegistryException,
             RemoteException;
```

The mapCertificate method is called to obtain a user name from an X.509 certificate chain that is supplied by the browser. The complete certificate chain is passed to this method and the implementation can validate the chain if needed and get the user information. A credential is created for this user for authorization purposes. If browser certificates are not supported in your configuration, you can create the CertificateMapNotSupportedException exception. The consequence of not supporting certificates is authentication failure if the challenge type is certificates, even if valid certificates are in the browser.

This method is called when certificates are provided for authentication. For Web applications, when the authentication constraints are set to CLIENT-CERT in the web.xml file of the application, this method is called to map a certificate to a valid user in the registry. For Java clients, this method is called to map the client certificates in the transport layer, when using transport layer authentication. When the identity assertion token, using the CSIv2 authentication protocol, is set to contain certificates, this method is called to map the certificates to a valid user.

In WebSphere Application Server Version 4.*x*, the input parameter is the X509Certificate certificate. In WebSphere Application Server Version 5.*x* and later, this parameter changes to accept an array of X509Certificate certificates such as a certificate chain. In Version 4.*x*, this parameter is called for Web applications only, but in version 5.*x* and later, you can call this method for both Web and Java clients.

```
public String getRealm()
      throws CustomRegistryException,
             RemoteException;
```

The getRealm method is called to get the name of the security realm. The name of the realm identifies the security domain for which the registry authenticates users. If this method returns a null value, a customRealm default name is used.

For the FileRegistrySample.java sample file, the getRealm method returns the customRealm string. One of the calls to this method occurs when the user registry information is validated. This method is the same method as in Version 4.x.

```
public Result getUsers(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

The getUsers method returns the list of users from the registry. The names of users depend on the pattern parameter. The number of users are limited by the limit parameter. In a registry that has many users, getting all the users is not practical. So the limit parameter is introduced to limit the number of users retrieved from the registry. A limit of zero (0) indicates to return all the users that match the pattern and might cause problems for large registries. Use this limit with care.

The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of asterisk (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object with a com.ibm.websphere.security.Result type . This object contains two attributes, a java.util.List and a java.lang.boolean attribute. The list contains the users that are returned and the Boolean flag indicates if more users are available in the user registry for the search pattern. This Boolean flag is used to indicate to the client whether more users are available in the registry.

In the `FileRegistrySample.java` sample file, the getUsers method retrieves the required number of users from the user registry and sets them as a list in the Result object. To find out if more users are presented than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to `true`. For pattern matching, the match method in the RegExpSample class is used, which supports wildcard characters such as the asterisk (*) and the question mark (?).

This method is called by the administrative console to add users to roles in the various map-users-to-roles panels. The administrative console uses the Boolean set in the Result object to indicate that more entries matching the pattern are available in the user registry.

In WebSphere Application Server Version 4.*x*, this method specifies to take only the pattern parameter. The return is a list. In WebSphere Application Server Version 5.*x* or later, this method is changed to take one additional parameter, the limit. Ideally, your implementation changes to take the limit value and limits the users that are returned. The return is changed to return a Result object, which consists of the list and a flag that indicates if more entries exist. When the list returns, use the Result.setList(List) method to set the list in the Result object. If more entries exist than requested in the limit parameter, set the Boolean attribute to `true` in the result object, using the Result.setHasMore method. The default for the Boolean attribute in the result object is `false`.

```
public String getUserDisplayName(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The getUserDisplayName method returns a display name for a user, if one exists. The display name is an optional string that describes the user that you can set in some registries. This descriptive name is for the user and does not have to be unique in the registry.

For example in Windows systems, you can display the full name of the user.

If you do not need display names in your registry, return null or an empty string for this method.

If display names existed for any user in WebSphere Application Server Version 4.*x*, these names were useful for the Enterprise JavaBean (EJB) method call getCallerPrincipal and the servlet calls getUserPrincipal and getRemoteUser. If the display names are not the same as the security name for any user, the display names are returned for the previously mentioned enterprise beans and servlet methods. Returning display names for these methods might become problematic in some situations because the display names might not be unique in the user registry. Avoid this problem by changing the default behavior to return the user security name instead of the user display name in this version of the product. For more information on how to set properties for the custom registry, see the section on *Setting Properties for Custom Registries*.

In the `FileRegistrySample.java` sample file, this method returns the display name of the user whose name matches the user name that is provided. If the display name does not exist, this method returns an empty string.

This method can be called by the product to present the display names in the administrative console, or by using the command line and the **wsadmin** tool. Use this method for display purposes only. This method is the same as in Version 4.*x*.

```
public String getUniqueUserId(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the unique ID of the user, given the security name.

In the `FileRegistrySample.java` sample file, this method returns the uniqueUserId value of the user whose name matches the supplied name. This method is called when forming a credential for a user and also when creating the authorization table for the application.

```
public String getUserSecurityName(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the security name of a user given the unique ID. In the `FileRegistrySample.java` sample file, this method returns the security name of the user whose unique ID matches the supplied ID.

This method is called to make sure a valid user exists for a given uniqueUserId. This method is called to get the security name of the user when the uniqueUserId is obtained from a token.

```
public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException,
             RemoteException;
```

This method indicates whether the given user is a valid user in the registry.

In the `FileRegistrySample.java` sample file, this method returns `true` if the user is found in the registry, otherwise this method returns `false`. This method is primarily called in situations where knowing if the user exists in the directory prevents problems later. For example, in the mapCertificate call, when the name is obtained from the certificate if the user is not found as a valid user in the user registry, you can avoid trying to create the credential for the user.

```
 public Result getGroups(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

The getGroups method returns the list of groups from the user registry. The names of groups depend on the pattern parameter. The number of groups is limited by the limit parameter. In a registry that has many groups, getting all the groups is not practical. So, the limit parameter is introduced to limit the number of groups retrieved from the user registry. A limit of zero (0) implies to return all the groups that match the pattern and can cause problems for large user registries. Use this limit with care. The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of asterisk (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of the com.ibm.websphere.security.Result type. This object contains the `java.util.List` and `java.lang.boolean` attributes. The list contains the groups that are returned and the Boolean flag indicates whether more groups are available in the user registry for the pattern searched. This Boolean flag is used to indicate to the client if more groups are available in the registry.

In the FileRegistrySample.java sample file, the getUsers method retrieves the required number of groups from the user registry and sets them as a list in the Result object. To find out if more groups are presented than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to true. For pattern matching, the match method in the RegExpSample class is used, which supports the asterisk (*) and question mark (?) characters.

This method is called by the administrative console to add groups to roles in the various map-groups-to-roles panels. The administrative console uses the boolean set in the Result object to indicate that more entries matching the pattern are available in the user registry.

In WebSphere Application Server Version 4, this method is used to take the pattern parameter only and returns a list. In WebSphere Application Server Version 5.x or later, this method is changed to take the limit parameter. Change to take the limit value and limit the users that are returned. The return is changed to return a Result object, which consists of the list and a flag that indicates whether more entries exist. Use the Result.setList(List) method to set the list in the Result object. If more entries exist than requested in the limit parameter, set the Boolean attribute to true in the Result object using the Result.setHasMore method. The default for the Boolean attribute in the Result object is false.

```
public String getGroupDisplayName(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The getGroupDisplayName method returns a display name for a group if one exists. The display name is an optional string that describes the group that you can set in some user registries. This name is a descriptive name for the group and does not have to be unique in the registry. If you do not need to have display names for groups in your registry, return null or an empty string for this method.

In the FileRegistrySample.java sample file, this method returns the display name of the group whose name matches the group name that is provided. If the display name does not exist, this method returns an empty string.

The product can call this method to present the display names in the administrative console or through the command line using the **wsadmin** tool. This method is used for display purposes only.

```
public String getUniqueGroupId(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the unique ID of the group that is given the security name.

In the FileRegistrySample.java sample file, this method returns the unique ID of the group whose name matches the supplied name. This method is called when creating the authorization table for the application.

```
public List getUniqueGroupIds(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the unique IDs of all the groups to which a user belongs.

In the FileRegistrySample.java sample file, this method returns the unique ID of all the groups that contain this uniqueUserID ID. This method is called when creating the credential for the user. As part of

creating the credential, all the groupUniqueIds IDsin which the user belongs are collected and put in the credential for authorization purposes when groups are given access to a resource.

```
public String getGroupSecurityName(String uniqueGroupId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the security name of a group given its unique ID.

In the `FileRegistrySample.java` sample file, this method returns the security name of the group whose unique ID matches the supplied ID. This method verifies that a valid group exists for a given uniqueGroupId ID.

```
public boolean isValidGroup(String groupSecurityName)
      throws CustomRegistryException,
             RemoteException;
```

This method indicates if the given group is a valid group in the registry.

In the `FileRegistrySample.java` sample file, this method returns `true` if the group is found in the registry, otherwise the method returns `false`. This method can be used in situations where knowing whether the group exists in the directory might prevent problems later.

```
public List getGroupsForUser(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns all the groups to which a user belongs whose name matches the supplied name. This method is similar to the getUniqueGroupIds method with the exception that the security names are used instead of the unique IDs.

In the `FileRegistrySample.java` sample file, this method returns all the group security names that contain the userSecurityName name.

This method is called by the administrative console or the scripting tool to verify that the users entered for the RunAs roles are already part of that role in the users and groups-to-role mapping. This check is required to ensure that a user cannot be added to a RunAs role unless that user is assigned to the role in the users and groups-to-role mapping either directly or indirectly through a group that contains this user. Because a group in which the user belongs can be part of the role in the users and groups-to-role mapping, this method is called to check if any of the groups that this user belongs to mapped to that role.

```
public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method retrieves users from the specified group. The number of users returned is limited by the limit parameter. A limit of zero (0) indicates to return all of the users in that group. This method is not directly called by the WebSphere Application Server security component. However, this method can be called by other components. In rare situations, if you are working with a user registry where getting all the users from any of your groups is not practical, you can create the NotImplementedException exception for the particular groups. In this case, verify that if the process choreographer is installed the staff assignments

are not modeled using these particular groups. If no concern exists about returning the users from groups in the user registry, it is recommended that you do not create the NotImplemented exception when implementing this method.

The return parameter is an object with a com.ibm.websphere.security.Result type. This object contains the java.util.List and java.lang.boolean attributes. The list contains the users that are returned and the Boolean flag, which indicates whether more users are available in the user registry for the search pattern. This Boolean flag indicates to the client whether users are available in the user registry.

In the example, this method gets one user more than the requested number of users for a group, if the limit parameter is not set to zero (0). If the method succeeds in getting one more user, the Boolean flag is set to `true`.

In WebSphere Application Server Version 4, this getUsers method is mandatory for the product. For WebSphere Application Server Version 5.x or later, this method can create the `NotImplementedException` exception in situations where it is not practical to get the requested set of users. However, create this exception in rare situations when as other components can be affected. In Version 4, this method accepts only the pattern parameter and returns a list. In Version 5, this method accepts the limit parameter. Change your implementation to take the limit value and limit the users that are returned. The return changes to return a Result object, which consists of the list and a flag that indicates whether more entries exist. When the list is returned, use the Result.setList(List) method to set the list in the Result object. If more entries than requested are in the limit parameter, set the Boolean attribute to `true` in the Result object using Result.setHasMore method. The default for the Boolean attribute in the Result object is `false`.

Attention: The first two lines of the following code sample are split for illustrative purposes only.

```
public com.ibm.websphere.security.cred.WSCredential
        createCredential(String userSecurityName)
     throws NotImplementedException,
            EntryNotFoundException,
            CustomRegistryException,
            RemoteException;
```

In this release the WebSphere Application Server, the createCredential method is not called. You can return *null*. In the example, a *null* value is returned.

Authentication protocol for EJB security

WebSphere Application Server Version 6.1 servers support the CSIv2 authentication protocol only. SAS is only supported between Version 6.0.x and earlier version servers that have been federated in a Version 6.1 cell. The option to select between SAS, CSIv2, or both is only available in the administration console when a Version 6.0.*x* or earlier release has been federated in a Version 6.1 cell.

V6.0.x SAS is the authentication protocol used by all previous releases of WebSphere Application Server and is maintained for backwards compatibility. The Object Management Group (OMG) has defined the authentication protocol called CSIv2 so that vendors can interoperate securely. CSIv2 is implemented in WebSphere Application Server with more features than SAS and is considered the strategic protocol.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Invoking Enterprise Java Beans (EJB) methods in a secure WebSphere Application Server environment requires an authentication protocol to determine the level of security and the type of authentication that occur between any given client and server for each request. It is the job of the authentication protocol during a method invocation to merge the server authentication requirements that are determined by the

object Interoperable Object Reference (IOR) with the client authentication requirements that are determined by the client configuration and come up with an authentication policy specific to that client and server pair.

The authentication policy makes the following decisions, among others, which are all based on the client and server configurations:
- What kind of connection can you make to this server--Secure Sockets Layer (SSL) or TCP/IP?
- If SSL is chosen, how strong is the encryption of the data?
- If SSL is chosen, do you authenticate the client using client certificates?
- Do you authenticate the client with a user ID and password? Does an existing credential exist?
- Do you assert the client identity to downstream servers?
- Given the configuration of the client and server, can a secure request proceed?

V6.0.x You can configure both protocols (SAS and CSIv2) to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client supports both and the server supports both, CSIv2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release) and the client supports both, the client chooses SAS for this request because the SAS protocol is what both have in common.

V6.0.x Choose a protocol by specifying the `com.ibm.CSI.protocol` property on the client side and configuring through the administrative console on the server side. More details are included in the SAS and CSIv2 properties articles.

Common Secure Interoperability Specification, Version 2

V6.0.x The Common Secure Interoperability Specification, Version 2 (CSIv2) defines the Security Attribute Service (SAS) that enables interoperable authentication, delegation, and privileges. The CSIv2 SAS and SAS protocols are entirely different. The CSIv2 SAS is a subcomponent of CSIv2 that supports SSL and interoperability with the EJB Specification, Version 2.1.

Security Attribute Service

V6.0.x The Common Secure Interoperability Specification, Version 2 Security Attribute Service (CSIv2 SAS) protocol is designed to exchange its protocol elements in the service context of a General Inter-ORB Protocol (GIOP) request and reply messages that are communicated over a connection-based transport. The protocol is intended for use in environments where transport layer security, such as that available through Secure Sockets Layer (SSL) and Transport Layer Security (TLS), is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality that might be applied to overcome corresponding deficiencies in an underlying transport. The CSIv2 SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports can be unified.

Connection and request interceptors

The authentication protocols that are used by WebSphere Application Server are add-on Interoperable Inter-ORB Protocol (IIOP) services. IIOP is a request-and-reply communications protocol that is used to send messages between two Object Request Brokers (ORBs). For each request made by a client ORB to a server ORB, an associated reply is made by the server ORB back to the client ORB. Prior to any request flowing, a connection between the client ORB and the server ORB must be established over the TCP/IP transport (SSL is a secure version of TCP/IP). The client ORB invokes the authentication protocol client connection interceptor, which is used to read the tagged components in the IOR of the object that is located on the server. As mentioned previously, the authentication policy is established here for the

request. Given the authentication policy (a coalescing of the server configuration with the client configuration), the strength of the connection is returned to the ORB. The ORB makes the appropriate connection, usually over SSL.

After the connection is established, the client ORB invokes the authentication protocol client request interceptor, which is used to send security information other than what is established by the transport. The security information includes the user ID and password token that are authenticated by the server, an authentication mechanism-specific token that is validated by the server, or an identity assertion token. Identity assertion is a way for one server to trust another server without the need to re-authenticate or re-validate the originating client. However, some work is required for the server to trust the upstream server. This additional security information is sent with the message in a *service context*. A service context has a registered identifier so that the server ORB can identify which protocol is sending the information.

V6.0.x The fact that a service context contains a unique identity is another way for WebSphere Application Server to support both SAS and CSIv2 simultaneously because both protocols have different service context IDs. After the client request interceptor finishes adding the service context to the message, the message is sent to the server ORB.

V6.0.x When the message is received by the server ORB, the ORB invokes the authentication protocol server request interceptor. This interceptor looks for the service context ID known by the protocol. When both SAS and CSIv2 are supported by a server, two different server request interceptors are invoked and both interceptors look for different service context IDs.

However, only one finds a service context for any given request. When the server request interceptor finds a service context, it reads the information in the service context. A method is invoked to the security server to authenticate or validate client identity. The security server either rejects the information or returns a credential. A credential contains additional information about the client that is retrieved from the user registry so that authorization can make the appropriate decision. Authorization is the process of determining if the user can invoke the request based on the roles that are applied to the method and the roles given to the user.

If a service context is not found by the CSIv2 server request interceptor, the interceptor process looks at the transport connection to see if a client certificate chain is sent. This process is done when SSL client authentication is configured between the client and server.

If a client certificate chain is found, the distinguished name (DN) is extracted from the certificate and is used to map to an identity in the user registry. If the user registry is Lightweight Directory Access Protocol (LDAP), the search filters defined in the LDAP registry configuration determine how the certificate maps to an entry in the registry. If the user registry is local OS, the first attribute of the distinguished name (DN) maps to the user ID of the registry. This attribute is typically the common name.

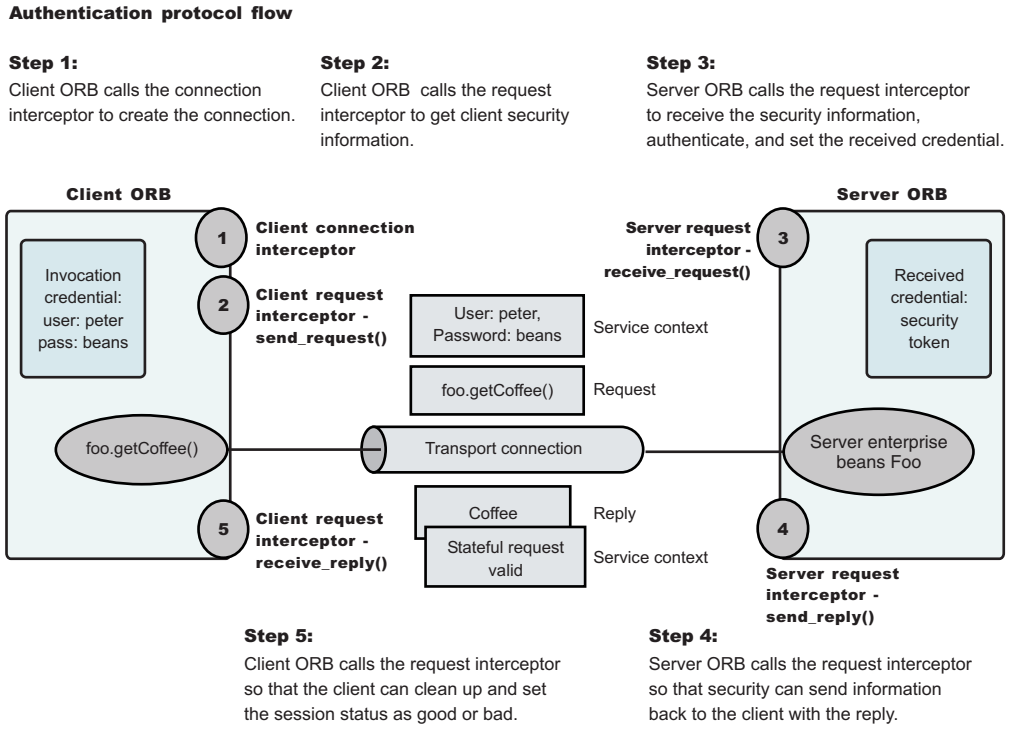
If the certificate does not map, no credential is created and the request is rejected. When valid security information is not presented, the method request is rejected and a NO_PERMISSION exception is sent back with the reply. However, when no security information is presented, an unauthenticated credential is created for the request and the authorization engine determines if the method gets invoked. For an unauthenticated credential to invoke an Enterprise JavaBean (EJB) method, either no security roles are defined for the method or a special Everyone role is defined for the method.

When the method invocation is completed in the EJB container, the server request interceptor is invoked again to complete server authentication and a new reply service context is created to inform the client request interceptor of the outcome. This process is typically for making the request *stateful*. When a stateful request is made, only the first request between a client and server requires that security

information is sent. All subsequent method requests need to send a unique context ID only so that the server can look up the credential that is stored in a session table. The context ID is unique within the connection between a client and server.

Finally, the method request cycle is completed by the client request interceptor receiving a reply from the server with a reply service context providing information so that the client-side stateful context ID can be confirmed and reused.

Specifying a stateful client is done through the property com.ibm.CSI.performStateful (true/false). Specifying a stateful server is done through the administrative console configuration.



. *Authentication protocol flow*

Authentication policy for each request

The authentication policy of a given request determines the security protection between a client and a server. A client or server authentication protocol configuration can describe required features, supported features, and non-supported features. When a client requires a feature, it can talk only to servers that either require or support that feature. When a server requires a feature, it can talk only to clients that either require or support that feature. When a client supports a feature, it can talk to a server that supports or requires that feature, but can also talk to servers that do not support the feature. When a server supports a feature, it can talk to a client that supports or requires the feature, but can also talk to clients that do not support the feature or chose not to support the feature.

For example, for a client to support client certificate authentication, some setup is required to either generate a self-signed certificate or to get one from a certificate authority (CA). Some clients might not need to complete these actions, therefore, you can configure this feature as not supported. By making this decision, the client cannot communicate with a secure server that requires client certificate authentication. Instead, this client can choose to use the user ID and password as the method of authenticating itself to the server.

Typically, supporting a feature is the most common way of configuring features. It is also the most successful during runtime because it is more forgiving than requiring a feature. Knowing how secure servers are configured in your domain, you can choose the right combination for the client to ensure successful method invocations and still get the most security. If you know that all of your servers support both client certificate and user ID and password authentication for the client, you might want to require one and not support the other. If both the user ID and password and the client certificate are supported on the client and server, both are performed, but user ID and password take precedence at the server. This action is based on the CSIv2 specification requirements.

Supported authentication protocols

Use this page to reference information regarding supported authentication protocols.

V6.0.x Beginning with WebSphere Application Server Version 6.1, the WebSphere Application Server Version 6.1 servers only support the Common Secure Interoperability Version 2 (CSIv2) authentication protocol. Secure Authentication Service (SAS) is only supported between Version 6.0.*x* and previous version servers that have been federated in a Version 6.1 cell. The option to select between SAS, CSIv2, or both will only be made available in the administration console when a Version 6.0.x or previous release has been federated in a Version 6.1 cell.

V6.0.x In future releases, IBM will no longer ship or support the Secure Authentication Service (SAS) IIOP security protocol. It is recommended that you use the Common Secure Interoperability version 2 (CSIv2) protocol.

V6.0.x You can configure both protocols to work simultaneously between Version 6.0.*x* and previous version servers that have been federated in a Version 6.1 cell. If a server supports both protocols, it exports an interoperable object reference (IOR) that contains tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client and server support both protocols, CSIv2 is used. However, if the server supports SAS (for example, the server is a previous WebSphere Application Server release) and the client supports both protocols, the client chooses SAS for this request.

Choose a protocol using the com.ibm.CSI.protocol property on the client side and configure this protocol through the administrative console on the server side.

Common Secure Interoperability Version 2 features

The following Common Secure Interoperability Version 2 (CSIv2) features are available in IBM WebSphere Application Server: Secure Sockets Layer (SSL) client certificate authentication, message layer authentication, identity assertion, and security attribute propagation.

- Identity Assertion.

  Supports a downstream server in accepting the client identity that is established on an upstream server, without having to authenticate again. The downstream server trusts the upstream server.
- Message Layer Authentication.

  Authenticates credential information and sends that information across the network so that a receiving server can interpret it.
- Security attribute propagation

  Supports the use of the authorization token to propagate serialized Subject contents and PropagationToken contents with the request. You can propagate these objects using a pure client or a server login that adds custom objects to the Subject. Propagating security attributes prevents downstream logins from having to make user registry calls to look up these attributes.

  Propagating security attributes is also useful when the security attributes contain information that is only available at the time of authentication. This information cannot be located using the user registry on downstream servers.

Identity assertion

Identity assertion is the invocation credential that is asserted to the downstream server.

When a client authenticates to a server, the received credential is set. When the authorization engine checks the credential to determine whether access is permitted, it also sets the *invocation* credential so that if the Enterprise JavaBeans (EJB) method calls another EJB method that is located on other servers, the invocation credential can be the identity used to invoke the downstream method. Depending on the RunAs mode for the enterprise beans, the invocation credential is set as the originating client identity, the server identity, or a specified different identity. Regardless of the identity that is set, when identity assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server identity, including the password or token, is sent in the client authentication token when basic authentication is enabled. The sending server identity is sent through a Secure Sockets Layer (SSL) client certification authentication when client certificate authentication is enabled. Basic authentication takes precedence over client certificate authentication.

Both identity tokens are needed by the receiving server to accept the asserted identity. The receiving server completes the following actions to accept the asserted identity:

- The server determines whether the sending server identity, sent with a basic authentication token or with an SSL client certificate, is on the trusted principal list of the receiving server. The server determines whether the sending server can send an identity token to the receiving server.
- After it is determined that the sending server is on the trusted list, the server authenticates the sending server to verify its identity.
- The server is authenticated by comparing the user ID and password from the sending server to the receiving server. If the credentials of the sending server are authenticated and on the trusted principal list, then the server proceeds to evaluate the identity token.

Evaluation of the identity token consists of the following four identity formats that exist in an identity token:

- Principal name
- Distinguished name
- Certificate chain
- Anonymous identity

The product servers that receive authentication information typically support all four identity types. The sending server decides which one is chosen, based on how the original client authenticated. The existing type depends on how the client originally authenticates to the sending server. For example, if the client uses Secure Sockets Layer (SSL) client authentication to authenticate to the sending server, then the identity token sent to the downstream server contains the certificate chain. With this information, the receiving server can perform its own certificate chain mapping and interoperability is increased with other vendors and platforms.

After the identity format is understood and parsed, the identity maps to a credential. For an ITTPrincipal identity token, this identity maps one-to-one with the user ID fields.

For an ITTDistinguishedName identity token, the mapping depends on the user registry. For Lightweight Directory Access Protocol (LDAP), the configured search filter determines how the mapping occurs. For LocalOS, the first attribute of the distinguished name (DN), which is typically the same as the common name, maps to the user ID of the registry.

Some user registry methods are called to gather additional credential information that is used by authorization. In a stateful server, this action completes once for the sending server and the receiving server pair where the identity tokens are the same. Subsequent requests are made through a session ID.

Identity assertion is only available using the Common Secure Interoperability Version 2 (CSIv2) protocol.

Identity assertions with trust validation

If you want an application or system provider to perform an identity assertion with trust validation, it can be accomplished by use of the Java Authentication and Authorization Service (JAAS) login framework, where trust validation is performed in one login module and credential creation in another. These two custom login modules are used to create a JAAS login configuration that performs a login to an identity assertion.

Two custom login module are required:

- A user-implemented trust association login module. This login module performs whatever trust verification the user requires. When trust is verified, the trust verification status and the login identity must be placed in a map in the share state of the login module to enable the credential creation login module to use that information. The map must be stored in the com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state property. State maps contain the following information:
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted – set to `true`, if trusted, and `false`, if not trusted.
  - com.ibm.wsspi.security.common.auth.module.IdenityAssertionLoginModule.principal – contains the principal of the identity.
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates – contains the certificate of the identity
- The com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule module performs the credential creation. It requires that the trust state information be in the login context's shared state. This login module is protected by the Java 2 security runtime permissions for the following:
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.initialize
  - com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.login

IdentityAssertionLoginModule searches for the trust information in the shared state property, com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state. This is a map that contains the trust status and the identity used to login. The map includes the following:

- com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted – if set to `true` it is trusted, `false` if not trusted.
- com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal – if a principal is used, it contains the principal of the identity necessary to login.
- com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates – if a certificate is used, it contains an array of a certificate chain that includes the identity necessary to login.

A WSLoginFailedException is returned if the state, trust, or identity information is missing. The login module then performs a login of the identity. The subject now contains the new identity.

Message layer authentication

Defines the credential information and sends that information across the network so that a receiving server can interpret it.

When you send authentication information across the network using a token the transmission is considered message layer authentication because the data is sent with the message inside a service context.

A pure Java client uses basic authentication, Generic Security Services Username Password (GSSUP), as the authentication mechanism to establish client identity.

However, a servlet can use either basic authentication (GSSUP) or the authentication mechanism of the server, Lightweight Third Party Authentication (LTPA), to send security information in the message layer. Use LTPA by authenticating or by mapping the basic authentication credentials to the security mechanism of the server.

The security token that is contained in a token-based credential is authentication mechanism-specific. The way that the token is interpreted is only known by the authentication mechanism. Therefore, each authentication mechanism has an object ID (OID) representing it. The OID and the client token are sent to the server, so that the server knows which mechanism to use when reading and validating the token. The following list contains the OIDs for each mechanism:

BasicAuth (GSSUP): oid:2.23.130.1.1.1
LTPA: oid:1.3.18.0.2.30.2
SWAM: No OID because it is not forwardable

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

On the server, the authentication mechanisms can interpret the token and create a credential, or they can authenticate basic authentication data from the client, and create a credential. Either way, the created credential is the *received* credential that the authorization check uses to determine if the user has access to invoke the method. You can specify the authentication mechanism by using the following property on the client side:

- com.ibm.CORBA.authenticationTarget

Basic authentication is currently the only valid value. You can configure the server through the administrative console.

Note: When **perform basic authentication** is enabled, if the client is not similarly configured (and does not pass a credential such as a user ID and password), the server object request broker (ORB) does not.

Configuring authentication retries

Situations occur where you want a prompt to display again if you entered your user ID and password incorrectly or you want a method to retry when a particular error occurs back at the client. If you can correct the error by information at the client side, the system automatically performs a retry without the client seeing the failure, if the system is configured appropriately.

Some of these errors include:
- Entering a user ID and password that are not valid
- Having an expired credential on the server
- Failing to find the stateful session on the server

By default, authentication retries are enabled and perform three retries before returning the error to the client. Use the com.ibm.CORBA.authenticationRetryEnabled property (`True` or `False`) to enable or disable authentication retries. Use the com.ibm.CORBA.authenticationRetryCount property to specify the number of retry attempts.

Configuring the Lightweight Third Party Authentication mechanism

You must configure Lightweight Third Party Authentication (LTPA) when you set up security for the first time. LTPA is the default authentication mechanism for WebSphere Application Server.

1. Open the administrative console.

   Type `http://`*`fully_qualified_host_name`*`:`*`port_number`*`/ibm/console` to access the administrative console in a Web browser.

   Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.

3. Select the appropriate group from the **Key set group** field that contains your public, private, and shared LTPA keys. These keys are used to encrypt and decrypt data that is sent between servers. You can access these key set group configurations using the Key set group link. In the Key set group configuration, you can indicate whether to automatically generate new keys and when to generate them.
4. Enter a positive integer value in the **Authentication cache timeout** field. This timeout value refers to how long an LTPA token is valid in minutes. The token contains this expiration time so that any server that receives the token can verify that the token is valid before proceeding further. When the token expires, the user must log in again. An optimal value for this field depends on your configuration. However, the default value is 10 minutes.
5. Enter a positive integer in the **Timeout value for forwarded credentials between servers** field. This value refers to how long the server credentials from another server are valid before they expire. The default value is 120 minutes. The value in the **Timeout value for forwarded credentials between servers** field must be greater than the value in the **Authentication cache timeout** field.
6. Click **Apply** or **OK**. The LTPA configuration is now set. Do not generate the LTPA keys in this step because they are automatically generated later. Proceed with the rest of the steps that are required to enable security, and start with single sign-on (SSO), if it is required.
7. Complete the information in the Security > Secure administration, applications, and infrastructure panel and click **OK**. The LTPA keys are generated automatically the first time. Do not generate the keys manually.

The previous steps configured LTPA.

After configuring LTPA, you can also complete the following tasks:
1. Generate key files. For more information, see “Generating Lightweight Third Party Authentication keys” on page 1239.
2. Export key files. For more information, see “Exporting Lightweight Third Party Authentication keys” on page 1240.
3. Import key files. For more information, see “Importing Lightweight Third Party Authentication keys” on page 1240.
4. Manage LPTA keys from multiple cells. For more information, see tsec_sslmanagelptakeys.dita.
5. If you are enabling security, you can also enable single sign-on (SSO). See:
   - “Configuring single sign-on capability with Tivoli Access Manager or WebSEAL” on page 1269
6. If you generated a new set of keys or imported a new set of keys, verify that the keys are saved to the master configuration by clicking **Save** at the top of the panel. Because LTPA authentication uses time-sensitive tokens, verify that the time, date, and time zone are synchronized among all of the product servers that are participating in the protected domain. Changes to the time, date, and time zone are done independently from WebSphere Application Server. If the clock skew is too high between servers, the LTPA token seems prematurely expired and causes authentication or validation failures.

Authentication mechanisms and expiration

Use this page to specify the shared keys and configure the authentication mechanism that is used to exchange information between servers. You can also use this page to specify the amount of time that the authentication information remains valid and specify the single sign-on configuration.

To view this administrative console page, complete the following steps:
1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Authentication mechanisms and expiration**.

After you configure the properties on this page, complete the following steps:
1. Click **Security > Secure administration, applications, and infrastructure**.
2. Verify that the appropriate registry is configured.

3. Click **Apply**. When security is enabled and any of these properties change, return to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

Key set group:

Specifies groups of public, private, and shared keys. These key groups enable the application server to manage multiple sets of Lightweight Third Party Authentication (LTPA) keys.

Generate Keys:

Specifies whether to generate a new set of LTPA keys in the configured keystore, and to update the runtime with the new keys. By default, LTPA keys are regenerated on a schedule every 90 days, configurable to the day of the week.

Each new set of LTPA keys is stored in the keystore associated with the key set group. A maximum number of keys (or even one) can be configured. However, it is recommended to have at least two keys; the old keys can be used for validation while the new keys are being distributed.

This step is not necessary during security enablement. A default set of keys is created during the first server startup. If any nodes are down during a key generation event, the nodes should be synchronized with the Deployment Manager before restart.

Authentication cache timeout:

Specifies the time period in minutes at which an LTPA token expires. Verify that this time period is less than the value for the Timeout value for forwarded credentials between servers field.

If the application server infrastructure security is enabled, the security cache timeout can influence performance. The timeout setting specifies how often to refresh the security-related caches. Security information pertaining to beans, permissions, and credentials is cached. When the cache timeout expires, all cached information not accessed within the timeout period is purged from the cache. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking a Lightweight Directory Access Protocol (LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance. Determine the best trade off for the application, by looking at usage patterns and security needs for the site.

The default security cache timeout value is 10 minutes. If you have a small number of users, it should be set higher than that, or if a large number of users, it should be set lower.

The LTPA timeout value should not be set lower than the security cache timeout. It is also recommended that the LTPA timeout value should be set higher than the orb request timeout value. However, there is no relation between the security cache timeout value and the orb request timeout value.

In a 20-minute performance test, setting the cache timeout so that a timeout does not occur yields a 40% performance improvement.

Data type	Integer
Units	Minutes and seconds
Default	10 minutes
Range:	Greater than 30 seconds

Timeout value for forwarded credentials between servers:

Specifies the period of time after which forwarded credentials expire.

Specify a value for this field that is greater than the authentication cache timeout value.

Default 120 minutes

Password:

Enter a password which will be used to encrypt and decrypt the LTPA keys from the SSO properties file. During import, this password should match the password used to export the keys at another LTPA server (for example, another application server Cell, Lotus Domino Server, and so on). During export, remember this password in order to provide it during the import operation.

After the keys are generated or imported, they are used to encrypt and decrypt the LTPA token. Whenever the password is changed, a new set of LTPA keys are automatically generated when you click **OK** or **Apply**. The new set of keys is used after the configuration changes are saved.

Data type String

Confirm password:

Specifies the confirmed password that is used to encrypt and decrypt the LTPA keys.

Use this password when importing these keys into other application server administrative domain configurations and when configuring SSO for a Lotus Domino server.

Data type String

Fully qualified key file name:

Specifies the name of the file that is used when importing or exporting keys.

Enter a fully qualified key file name, and click **Import Keys** or **Export Keys**.

Data type String

Internal server ID:

Specifies the server ID that is used for interprocess communication between servers. The server ID is protected with an LTPA token when sent remotely. You can edit the internal server ID to make it identical to server IDs across multiple application server administrative domains (cells). By default this ID is the cell name.

This internal server ID should only be used in a Version 6.1 or higher environment. For mixed-version Cells, you should convert to using a server user ID and server password for interoperability.

To switch back to the server user ID and password for interoperability, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select a user registry, and click **Configure**.
3. Select the **Server identity that is stored in the repository** option and type a valid registry ID and password.

Data type String

Import Keys:

Specifies whether the server imports new LTPA keys.

To support single sign-on (SSO) in the application server product across multiple application server domains (cells), share the LTPA keys and the password among the domains. You can use the **Import Keys** option to import the LTPA keys from other domains. The LTPA keys are exported from one of the cells to a file. To import a new set of LTPA keys, complete the following steps:

1. Enter the appropriate password in the Password and Confirm password fields.
2. Click **OK** and click **Save**.
3. Enter the directory location where the LTPA keys are located in the Fully qualified key file name field prior to clicking **Import keys**.
4. Do not click **OK** or **Apply**, but save the settings.

Export Keys:

Specifies whether the server exports LTPA keys.

To support single sign-on (SSO) in the WebSphere product across multiple application server domains (cells), share the LTPA keys and the password among the domains. Use the Export Keys option to export the LTPA keys to other domains.

To export the LTPA keys, make sure that the system is running with security enabled and is using LTPA. Enter the file name in the Fully qualified key file name field and click **Export Keys**. The encrypted keys are stored in the specified file.

Use SWAM-no authenticated communication between servers:

Specifies the Simple WebSphere Authentication Mechanism (SWAM). Unauthenticated credentials are forwarded between servers. When a caller process invokes a remote method, its identity is not verified. Depending upon the security permissions for the EJB methods, authentication failures might occur.

SWAM is a deprecated feature and will be removed in a future release. It is recommend that you use LTPA for authenticated communication between servers.

Generating Lightweight Third Party Authentication keys

WebSphere Application Server generates Lightweight Third Party Authentication (LTPA) keys automatically during the first server startup. You can generate additional keys as you need them in the Authentication mechanisms and expiration panel.

Complete the following steps to generate new LTPA keys in the administrative console.

1. Access the administrative console.

   Type `http://`*`fully_qualified_host_name`*`:`*`port_number`*`/ibm/console` to access the administrative console in a Web browser.
2. Verify that all the WebSphere Application Server processes are running, including the cell, nodes, and application servers.

   Important: If any of the servers are down at the time of key generation and then restarted later, these servers might contain old keys. Copy the new set of keys to these servers to restart them after you generate them.
3. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
4. Click **Generate keys** to generate a new set of LTPA keys in the local keystore and update the runtime with the new keys. By default, LTPA keys are regenerated on a schedule every 90 days, configurable

to the day of the week. Each new set of LTPA keys is stored in the keystore that is associated with the key set group. The same password that is already stored in the configuration is used when you generate new keys.

Tip: This step is not necessary when you enable security because, by default, a set of keys is created during the first server startup. However, the keystore should have at least two keys: the old keys can be used for validation while the new keys are being distributed. If any nodes are down during a key generation event, the nodes should be synchronized with the Deployment Manager before restarting the server.

5. Restart the server for the changes to become active.

After WebSphere Application Server generates and saves a new set of keys, the generated keys are not used in the configuration until WebSphere Application Server is restarted. Token generation uses the keys that were last imported. To view the latest key version, see “Activating Lightweight Third Party Authentication key versions” on page 1242.

Exporting Lightweight Third Party Authentication keys

To support single sign-on (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, you must share the Lightweight Third Party Authentication (LTPA) keys and the password among the domains.

Make sure that the time in the domains is similar so that you do not mistakenly interpret the tokens as expired between the cells.

Complete the following steps in the administrative console to export key files for LTPA so that they can be shared across domains:

1. Type `http://server_name:port_number/ibm/console` in a Web browser to access the administrative console.
2. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
3. In the Password and Confirm password fields, enter the password that is used to encrypt the LTPA keys. Remember the password so that you can use it later when the keys are imported into the other cell.
4. In the Fully qualified key file name field, specify the fully qualified path to the location where you want the exported LTPA keys to reside. You must have write permission to this file.
5. Click **Export keys** to export the keys to the location that you specified in the **Fully qualified key file name** field.
6. Specify the **Internal server ID** that is used for interprocess communication between servers. The server ID is protected with an LTPA token when sent remotely. You can edit the internal server ID to make it identical to server IDs across multiple application server administrative domains (cells). By default this ID is the cell name.
7. Click **OK** and **Save**.

You can share LTPA keys and passwords among domains on WebSphere Application Server.

After exporting the keys from one cell, you must import those keys into the other cell. For more information, see “Importing Lightweight Third Party Authentication keys”

Importing Lightweight Third Party Authentication keys

To support single sign-on (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, you must share the LTPA keys and the password among the domains. You can import LTPA keys from other domains and export keys to other domains.

After you export LTPA keys from one cell, you must import these keys into another cell. To import keys, you must know the password for the exported key file to access the LTPA keys. Verify that key files are exported from one of the cells into a file.

Complete the following steps in the administrative console to import key files for LTPA.

1. Access the administrative console for the cell that will receive the imported keys by typing `http://server_name:port_number/ibm/console` in a Web browser.
2. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
3. In the **Password** and **Confirm password** fields, enter the password that is used to decrypt the LTPA keys . This password must match the password that was used in the cell from which you are importing the keys.
4. In the **Fully qualified key file name** field, specify the fully qualified path to the location where the signer keys reside. You must have write permission to this file.
5. Click **Import keys** to import the keys to the location that you specified in the **Fully qualified key file name** field.
6. Click **OK** and **Save** to save the changes to the master configuration. It is important to save the new set of keys to match the new password so that no problems are encountered when starting the servers later.

After a new set of keys is generated and saved, the generated keys are not used in the configuration until WebSphere Application Server is restarted.

Important: After you enter the password in the Password and Confirm password fields and click **Save**, the password is not redisplayed on the administrative console panel.

Disabling automatic generation of Lightweight Third Party Authentication keys

You can disable the automatic generation of new Lightweight Third Party Authentication (LTPA) keys for key sets that are members of a key set group. Automatic generation creates new keys on a schedule that you specify when you configure a key set group, which manages one or more key sets. WebSphere Application Server uses key set groups to automatically generate cryptographic keys or multiple synchronized key sets.

You must know the name of the key set group and the management scope where the key set group is defined.

LTPA keys are used to encrypt the LTPA token. You might want to disable the auto-generation of these keys so that you can generate them on a schedule. The following steps are needed to complete this task in the administrative console.

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations**.
2. Expand the tree to the inbound or outbound management scope that contains the key set group, and then click the scope link.
3. Under Related Items, click **Key Set Groups**.
4. Click the key set group that you want to disable.
5. Clear the **Automatically generate keys** option.
6. Click **OK** and **Save** to save the changes to the master configuration.
7. Start the server again for the changes to become active.

You have disabled the automatic generation of LTPA keys for the key sets in the key set group.

Tip: You can generate keys manually at any time by completing the following steps:

1. Open the key set group collection.

2. Select the check box beside the key set group.
3. Click **Generate keys**.

Managing LTPA keys from multiple WebSphere Application Server cells

You can specify the shared keys and configure the authentication mechanism that is used to exchange information between servers to import and export LTPA keys across multiple WebSphere Application Server cells.

You must be sure that the exported key file for the multiple cells is accessible on the host where WebSphere Application Server is running. Also, you must know the password that was used when the keys were exported.

Complete the following steps to manage LTPA keys using the administrative console.

1. Access the administrative console.

   Type `http://fully_qualified_host_name:port_number/ibm/console` to access the administrative console in a Web browser.
2. Verify that all of the WebSphere Application Server processes are running, including cells, nodes, and all of the application servers. If any of the servers are down at the time of key generation and then brought back up later, these servers might contain old keys. Copy the new set of keys to these servers, then bring them back up.
3. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
4. Type the password for the LTPA keys in the **Password** field. Enter a password that is used to encrypt and decrypt the LTPA keys from the single sign-on (SSO) properties file. During import, this password should match the password that is used to export the keys at another LTPA server. During export, remember this password in order to provide it during the import operation.
5. Type the password again in the **Confirm password** field.
6. Select from among the following options:
   - To support SSO in the WebSphere product across multiple application server domains (cells), you can share the LTPA keys and the password among the domains. Before exporting, make sure that security is enabled and using LTPA on the system that is running. For more information, see “Exporting Lightweight Third Party Authentication keys” on page 1240.
   - To support SSO in the application server product across multiple application server domains (cells), you can share the LTPA keys and the password among the domains. For more information, see “Importing Lightweight Third Party Authentication keys” on page 1240.
   - To import LTPA keys for the current cell if they were previously exported, see “Importing Lightweight Third Party Authentication keys” on page 1240.
7. Start the server again for any changes you make to become active.

The shared LTPA keys are now available for WebSphere Application Server to use for secure connections.

After the keys are generated or imported, they are used to encrypt and decrypt the LTPA token. To view the latest key version, see “Activating Lightweight Third Party Authentication key versions.”

Activating Lightweight Third Party Authentication key versions

Key sets manage Lightweight Third Party Authentication (LTPA) keys in a key store that is based on a key alias prefix. A key alias prefix is automatically generated when you generate a new key and store it in a key store. Key stores can contain multiple versions of keys for any given key alias prefix. You can specify a maximum number of active keys in the key set configuration.

You must know the name of the key set group and the management scope where the key set group is defined. Complete the following steps in the administrative console.

LTPA keys are used to encrypt the LTPA token. You might want to set a specific number of active keys that WebSphere Application Server returns when the server queries for keys for a particular key set. The following steps are needed to complete this task in the administrative console.

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations**.
2. Expand the tree to the inbound or outbound management scope that contains the key set group, and then click the scope link.
3. Under Related Items, click **Key Sets**.
4. Click the key set that you want to modify.
5. In the **Maximum number of keys referenced** field, type a numerical value for the maximum number of keys that you want to activate.
6. Click **OK** and **Save** to save the changes to the master configuration.
7. Start the server again for the changes to become active. WebSphere Application Server activates only the number of recent keys that you specified.

The **Maximum number of keys referenced** value determines how many active keys are returned when the server queries for keys for the selected key set.

You can click **Active key history** in the Key set panel to display the keys that are active for this key set.

Integrating third-party HTTP reverse proxy servers

These steps are required to use either a WebSEAL trust association interceptor or your own trust association interceptor with a reverse proxy security server.

WebSphere Application Server enables you to use multiple trust association interceptors. The Application Server uses the first interceptor that can handle the request.

1. Access the administrative console.

   Type `http://`*`fully_qualified_host_name`*`:`*`port_number`*`/ibm/console` in a Web browser.

   Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security > Secure administration, applications, and infrastructure**.
3. Under Web security, click **Trust association**.
4. Select the **Enable trust association** option.
5. Under Additional properties, click **Interceptors**. The default value appears.
6. Verify that the appropriate trust association interceptors are listed. If you need to use a WebSEAL trust association interceptor, see “Configuring single sign-on using the trust association interceptor” on page 1277 or “Configuring single sign-on using trust association interceptor ++” on page 1278. If you are not using WebSEAL and need to use a different interceptor, complete the following steps:
   a. Select both the **com.ibm.ws.security.web.WebSealTrustAssociationInterceptor** and the **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** class name and click **Delete**.
   b. Click **New** and specify a trust association interceptor.

Enables trust association.

1. If you are enabling security, make sure that you complete the remaining steps for enabling security.
2. Save, stop and restart all of the product servers (deployment managers, nodes and Application Servers) for the changes to take effect.

Trust association settings

Use this page to enable trust association, which integrates application server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the proxy server.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand Web security and click **Trust association**.

When security is enabled and any of these properties change, go to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

Enable trust association:

Specifies whether trust association is enabled.

Data type:	Boolean
Default:	Disable
Range:	Enable or Disable

Trust association interceptor collection

Use this page to specify trust information for reverse security proxy servers.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand Web security and click **Trust association**.
3. Under Additional Properties, click **Interceptors**.

When security is enabled and any of these properties are changed, go to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

Interceptor class name:

Specifies the trust association interceptor class name.

Data type
String

Default
`com.ibm.ws.security.web.WebSealTrustAssociationInterceptor`

Trust association interceptor settings

Use this page to specify trust information for reverse security proxy servers.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand Web security and click **Trust association**.
3. Under Additional Properties, click **Interceptors > New**.

When security is enabled and any of these properties are changed, go to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

Interceptor class name:

Specifies the trust association interceptor class name.

Data type
String

Default
`com.ibm.ws.security.web.WebSealTrustAssociationInterceptor`

Implementing single sign-on to minimize Web user authentications

With single sign-on (SSO) support, Web users can authenticate once when accessing Web resources across multiple WebSphere Application Servers. Form login mechanisms for Web applications require that SSO is enabled. Use this topic to configure single sign-on for the first time.

SSO is supported only when Lightweight Third Party Authentication (LTPA) is the authentication mechanism.

When SSO is enabled, a cookie is created containing the LTPA token and inserted into the HTTP response. When the user accesses other Web resources in any other WebSphere Application Server process in the same domain name service (DNS) domain, the cookie is sent in the request. The LTPA token is then extracted from the cookie and validated. If the request is between different cells of WebSphere Application Servers, you must share the LTPA keys and the user registry between the cells for SSO to work. The realm names on each system in the SSO domain are case sensitive and must match identically.

Windows For local OS, the realm name is the domain name if a domain is in use. If a domain is not used, the realm name is the machine name.

Linux The realm name is the same as the host name.

For the Lightweight Directory Access Protocol (LDAP) the realm name is the host:port realm name of the LDAP server. The LTPA authentication mechanism requires that you enable SSO if any of the Web applications have form login as the authentication method.

Because single sign-on is a subset of LTPA, it is recommended that you read “Lightweight Third Party Authentication” on page 1193 for more information.

When you enable security attribute propagation, the following cookies are added to the response:

LtpaToken
LtpaToken is used for inter-operating with previous releases of WebSphere Application Server. This token contains the authentication identity attribute only.

LtpaToken2
LtpaToken2 contains stronger encryption and enables you to add multiple attributes to the token. This token contains the authentication identity and additional information such as the attributes that are used for contacting the original login server and the unique cache key for looking up the Subject when considering more than just the identity in determining uniqueness.

For more information, see “Security attribute propagation” on page 1206.

Note: LtpaToken is generated for releases prior to WebSphere Application Server Version 5.1.1. LtpaToken2 is generated for WebSphere Application Server Version 5.1.1 and beyond.

Token type	Purpose	How to specify
LtpaToken only	This token type is used for the same SSO behavior existing in WebSphere Application Server Version 5.1 and previous releases. Also, this token type is interoperable with those previous releases.	Disable the **Web inbound security attribute propagation** option, which is located in the SSO configuration panel in the administrative console. To access this panel, complete the following steps: 1. Click **Security > Secure administration, applications, and infrastructure**. 2. Under Web security, click **Single sign-on (SSO)**.
LtpaToken2 only	This token type is used for Web inbound security attribute propagation and uses the AES, CBC, PKCS5 padding encryption strength (128-bit key size). However, this token type is not interoperable with releases prior to WebSphere Application Server Version 5.1.1. The token type supports multiple attributes that are specified in the token, mostly containing information to contact the original login server.	Enable the **Web inbound security attribute propagation** option in the SSO configuration panel within the administrative console. Disable the **Interoperability mode** option in the SSO configuration panel within the administrative console. To access this panel, complete the following steps: 1. Click **Security > Secure administration, applications, and infrastructure**. 2. Under Web security, click **Single sign-on (SSO)**.
LtpaToken and LtpaToken2	These tokens together support both of the previous two options. The token types are interoperable with releases prior to WebSphere Application Server Version 5.1.1 because LtpaToken is present. The security attribute propagation function is enabled because the LtpaToken2 is present.	Enable the **Web inbound security attribute propagation** option in the SSO configuration panel within the administrative console. Enable the **Interoperability mode** option in the SSO configuration panel within the administrative console. To access this panel, complete the following steps: 1. Click **Security > Secure administration, applications, and infrastructure**. 2. Under Web security, click **Single sign-on (SSO)**.

The following steps are required to configure SSO for the first time.

1. Open the administrative console.

   Type `http://localhost:`*`port_number`*`/ibm/console` to access the administrative console in a Web browser.

   Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security > Secure administration, applications, and infrastructure**.
3. Under Web security, click **Single sign-on (SSO)**.
4. Click the **Enabled** option if SSO is disabled. After you click the **Enabled** option, make sure that you complete the remaining steps to enable security.
5. Click **Requires SSL** if all of the requests are expected to use HTTPS.

6. Enter the fully qualified domain names in the **Domain name** field where SSO is effective. If you specify domain names, they must be fully qualified. If the domain name is not fully qualified, WebSphere Application Server does not set a domain name value for the LtpaToken cookie and SSO is valid only for the server that created the cookie.

   When you specify multiple domains, you can use the following delimiters: a semicolon (**;**), a space ( ), a comma (**,**), or a pipe (**|**). WebSphere Application Server searches the specified domains in order from left to right. Each domain is compared with the host name of the HTTP request until the first match is located. For example, if you specify `ibm.com; austin.ibm.com` and a match is found in the ibm.com domain first, WebSphere Application server does not continue to search for a match in the austin.ibm.com domain. However, if a match is not found in either the ibm.com or austin.ibm.com domains, then WebSphere Application Server does not set a domain for the LtpaToken cookie.

   You can configure the Domain name field using any of the following values:

Domain name value type	Example	Purpose
Blank		The domain is not set. This causes the browser to set the domain to the request host name. The sign-on is valid on that single host only.
Single domain name	austin.ibm.com	If the request is to a host within the configured domain, the sign-on is valid for all hosts within that domain. Otherwise, it is valid on the request host name only.
UseDomainFromURL	UseDomainFromURL	If the request is to a host within the configured domain, the sign-on is valid for all hosts within that domain. Otherwise, it is valid on the request host name only.
Multiple domain names	austin.ibm.com;raleigh.ibm.com	The sign-on is valid for all hosts within the domain of the request host name.
Multiple domain names and UseDomainFromURL	• austin.ibm.com;raleigh.ibm.com; UseDomainFromURL	The sign-on is valid for all hosts within the domain of the request host name.

   If you specify the UseDomainFromURL, WebSphere Application Server sets the SSO domain name value to the domain of the host that makes the request. For example, if an HTTP request comes from server1.raleigh.ibm.com, WebSphere Application Server sets the SSO domain name value to `raleigh.ibm.com` .

   Tip: The value, `UseDomainFromURL`, is case insensitive. You can type `usedomainfromurl` to use this value.

   For more information, see “Single sign-on settings” on page 1269.
7. **Optional:** Enable the **Interoperability mode** option if you want to support SSO connections in WebSphere Application Server version 5.1.1 or later to interoperate with previous versions of the application server. This option sets the old-style LtpaToken token into the response so it can be sent to other servers that work only with this token type. However, this option applies only when the **Web inbound security attribute propagation** option is enabled. In this case, both the LtpaToken and LtpaToken2 tokens are added to the response. Otherwise, only the LtpaToken2 token is added to the response. If the **Web inbound security attribute propagation** option is disabled, then only the LtpaToken token is added to the response.
8. **Optional:** Enable the **Web inbound security attribute propagation** option if you want information added during the login at a specific front-end server to propagate to other front-end servers. The SSO token does not contain any sensitive attributes, but does understand where the original login server exists in cases where it needs to contact that server to retrieve serialized information. It also contains

the cache look-up value for finding the serialized information in DynaCache, if both front-end servers are configured in the same DRS replication domain. For more information, see “Security attribute propagation” on page 1206.

Important: If the following statements are true, it is recommended that you disable the **Web inbound security attribute propagation** option for performance reasons:

- You do not have any specific information added to the Subject during a login that cannot be obtained at a different front-end server.
- You did not add custom attributes to the PropagationToken token using WSSecurityHelper application programming interfaces (APIs).

If you find that you are missing custom information in the Subject, re-enable the **Web inbound security attribute propagation** option to see if the information is propagated successfully to other front-end application servers. If you disable SSO, but use a trust association interceptor instead, you might still need to enable the **Web inbound security attribute propagation** option if you want to retrieve the same Subject generated at different front-end servers.

9. Click **OK**.

For the changes to take effect, save, stop, and restart all the product servers.

Configuring single sign-on capability with SPNEGO TAI

WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server. To deploy and use the SPNEGO TAI you need to examine your installation and decide on how best to configure the SPNEGO TAI.

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. However, you may need to configure LTPA prior to configuring the SPNEGO TAI. LTPA is the required authentication mechanism for all trust association interceptors. You can configure LTPA by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.

Note: Enabling Web security single sign-on (SSO) is optional when you configure the SPNEGO TAI. For more information, see “Implementing single sign-on to minimize Web user authentications” on page 1245.

Answer the following questions to establish how the SPNEGO TAI is deployed.

1. What is your criteria for intercepting HTTP requests?
   - You must decide if the SPNEGO TAI deployment will use the HTTPHeaderFilter class as the default. If you do use this class, then you must specify the exact filter properties for this class. The default behavior of the SPNEGO TAI is to use the com.ibm.ws.spnego.HTTPHeaderFilter class to intercept all requests.
   - If you do not use the sample com.ibm.ws.spnego.HTTPHeaderFilter class, then you must define a new class that implements the com.ibm.wsspi.security.spnego.SpnegoTAIFilter interface.
   - You can decide to further control what HTTP requests are intercepted using the Service Provider Programming Interface (SPI), “Filtering HTTP requests for SPNEGO TAI” on page 1267

   See “SPNEGO TAI custom configuration attributes” on page 1254 for descriptions of
   - com.ibm.ws.security.spnego.SPN<id>.filterClass
   - com.ibm.ws.security.spnego.SPN<id>.filter
2. Is user Id mapping to be used? If not, why not? WebSphere Application Server enables you to define or develop a custom login module to map user IDs. See “Mapping user Ids from client to server for SPNEGO” on page 1267 for more detail about performing this mapping.

You must decide, before deploying the TAI, whether or not to use this custom login module to perform the SPNEGO TAI identity mapping

3. What type of encryption is to be used to process the SPNEGO tokens? Microsoft Windows Active Directory supports two different Kerberos encryption types: RC4-HMAC and DES-CBC-MD5. The IBM Java Generic Security Service (JGSS) library (and SPNEGO library) support both of these encryption types.

   Restriction: RC4-HMAC encryption is only supported with a Windows 2003 Server key distribution center (KDC). RC4-HMAC encryption is not supported when using a Windows 2000 Server as a Kerberos KDC.

4. How will you handle credential delegation? Kerberos supports the delegation of credentials. A server that receives Kerberos credentials from a client can impersonate that client to other servers by using delegated credentials. Since SPNEGO TAI tokens are a wrapping of a Kerberos credential, a server that receives Kerberos credentials within an SPNEGO token can use those Kerberos credentials to impersonate the original user. That server can interact using SPNEGO over HTTP as a SPNEGO client to other SPNEGO servers by composing an appropriate HTTP Authorization header.
5. Will the SPNEGO TAI be deployed in a single or multiple domain name service (DNS) domain environment?

   Web browsers running on Windows are sensitive to DNS domains. They only send a SPNEGO token when the target host name identifies a host name defined in the DNS domain of the client machine. You can use HTTP redirection to support this configuration with the creation of a pseudo Kerberos service principal name (SPN) in each DNS domain. All SPNs that WebSphere Application Server supports must have their secret keys available in Kerberos `keytab` files. To enable single sign-on across multiple DNS domains, a separate Kerberos keytab file is generated for each SPN per domain. These individual Kerberos keytab files must be merged before they can be used by WebSphere Application Server.
6. How frequently will application servers reload the SPNEGO TAI properties The SPNEGO TAI has an optional property reload feature that allows the reloading of the TAI properties without restarting the Java virtual machine (JVM). This reload feature is controlled by the system properties `com.ibm.ws.security.spnego.propertyReloadFile` and `com.ibm.ws.security.spnego.propertyReloadTimeout`. These properties taken together enable the SPNEGO TAI internal properties to be reloaded from a file on the file system after a certain time period. If the com.ibm.ws.security.spnego.propertyReloadTimeout attribute is set to a valid integer value, and the com.ibm.ws.security.spnego.propertyReloadFile attribute points to a file on the file system, then each JVM reloads the SPNEGO TAI properties from the file after the timeout period expires. Also, the SPNEGO TAI properties are reloaded only if the date on the file has changed. If these reload properties are not set, then the SPNEGO TAI properties are only loaded once, at JVM initialization, from the SPNEGO TAI custom properties that are defined in WebSphere Application Server configuration data. See “SPNEGO TAI JVM configuration attributes” on page 1264 for more information about these reload properties.

The Windows Active Directory (Web) administrator, the WebSphere Application Server administrator, and the application team review and answer these questions to determine the best deployment and configuration settings for the SPNEGO TAI.

Configuring WebSphere Application Server environment to use SPNEGO:

The objective of the Web administrator is to configure and administer the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) in WebSphere Application Server to provide users who successfully authenticated in a Microsoft Active Directory domain with a single sign-on capability. The administrator specifies additional criteria that selects what Web transactions to authenticate in the single sign-on environment.

Verify that the Web browser is configured to use the SPNEGO authentication mechanism. “Configuring the Web browser to use SPNEGO” on page 1268 describes what the user needs to do to configure the Web browser.

The Web administrator configures and enables the SPNEGO TAI. The process to configure and enable the SPNEGO TAI operation in WebSphere Application Server requires some tasks that use tools not supplied by WebSphere Application Server. Refer to those appropriate documents that describe these tasks and tools. These documents are supplied by the appropriate supplier.

1. Create a user account in the Microsoft Active Directory.
2. Map the user account to the Kerberos service principal name (SPN). This user account represents the WebSphere Application Server as being a Kerberize’d service with the Kerberos key distribution center (KDC). Use the **setspn** tool to establish WebSphere Application Server as the user. This user account is not the account name of the user. More information about the **setspn** tool can be found here, Windows 2003 Technical Reference (setspn command)
3. Create the Kerberos keytab file and make it available to WebSphere Application Server. Use the **ktpass** tool to create the Kerberos keytab file (`krb5.keytab`). Windows 2003 Technical Reference (Kerberos keytab file and ktpass command) provides more information on creating the Kerberos keytab file.
4. Configure and enable the application server and the associated SPNEGO TAI using the administrative console or using the wsadmin command to perform command tasks. See “Configuring SPNEGO TAI in WebSphere Application Server” on page 1253.
5. Select Lightweight Third-Party Authentication (LTPA) as the authentication mechanism. See “Configuring the Lightweight Third Party Authentication mechanism” on page 1235.
6. Enable the SPNEGO TAI in each application server in which it is defined.
7. Install the Kerberos keytab file (`krb5.keytab`).
8. Update the associated Kerberos configuration (`krb5.conf`).
9. Configure JVM properties and enable SPNEGO TAI. See “Configuring JVM properties and enabling SPNEGO TAI in WebSphere Application Server” on page 1263.

WebSphere Application Server is configured to use the SPNEGO TAI.

Configuring the Kerberos configuration properties:

The Kerberos configuration properties, or `krb5.ini` and `krb5.conf` files, must be configured on every WebSphere Application Server instance in a cell in order to use the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

A configuration file for a Kerberos configuration on a UNIX platform follows:

```
[libdefaults]
 default_realm = WSSEC.AUSTIN.IBM.COM
        default_keytab_name = FILE:/etc/krb5.keytab
        default_tkt_enctypes = des-cbc-md5
        default_tgs_enctypes = des-cbc-md5
[realms]
        WSSEC.AUSTIN.IBM.COM = {
  kdc = axel.austin.ibm.com:88
              default_domain = austin.ibm.com
  admin_server = axel.austin.ibm.com
              default domain = austin.ibm.com
        }
[domain_realm]
        .austin.ibm.com = WSSEC.AUSTIN.IBM.COM
```

In the above example, the Kerberos key distribution center (KDC) is `axel.austin.ibm.com:88`. The Kerberos keytab file is located at: `FILE:/etc/krb5.keytab`. The Kerberos realm name is

WSSEC.AUSTIN.IBM.COM, which is also the Microsoft domain controller. After you update your Kerberos configuration properties for your particular system deployment, your Kerberos configuration is ready for use with the SPNEGO TAI.

The Kerberos configuration is configured for use with the SPNEGO TAI.

Creating the Kerberos configuration file for use with the SPNEGO TAI:

You use the wsadmin utility to create the Kerberos keytab configuration file for use with the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Note: A Kerberos keytab configuration file contains a list of keys that are analogous to user passwords. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk, which makes them readable only by authorized users.

Use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.
3. At the **wsadmin** prompt, enter the following command:

   $AdminTask createKrbConfigFile

   You can use the following parameters with this command:

Option	Description
<krbPath>	This parameter is required. It provides the fully qualified file system location of the Kerberos configuration (krb5.ini or krb5.conf) file.
<realm>	This parameter is required. It provides the Kerberos realm name. The value of this attribute is used by the SPNEGO TAI to form the Kerberos service principal name for each of the hosts specified with the property com.ibm.ws.security.spnego.SPN<id>.hostName.
<kdcHost>	This parameter is required. It provides the host name of the Kerberos Key Distribution Center (KDC).
<kdcPort>	This parameter is optional. It provides the port number of the KDC. The default value, if not specified, is 88.
<dns>	This parameter is required. It provides the default domain name service (DNS) that is used to produce a fully qualified host name.
<keytabPath>	This parameter is required. It provides the file system location of the Kerberos keytab file.
<encryption>	This parameter is optional. It identifies the list of supported encryption types, separated by a space. The specified value is used for the default_tkt_enctypes and default_tgs_enctypes. The default encryption types, if not specified, are des-cbc-md5 and rc4-hmac.

   In the following example, the wsadmin command creates the krb5.ini file in the c:\winnt directory. The default Kerberos keytab file is also in c:\winnt. The actual Kerberos realm name is WSSEC.AUSTIN.IBM.COM and the KDC host name is host1.austin.ibm.com.

```
wsadmin>$AdminTask createKrbConfigFile {-krbPath
c:\winnt\krb5.ini -realm WSSEC.AUSTIN.IBM.COM -kdcHost host1.austin.ibm.com
 -dns austin.ibm.com -keytabPath c:\winnt\krb5.keytab}
```

The Kerberos keytab configuration file is created for use with the SPNEGO TAI.

Note: The default Kerberos `krb5.ini` file on Windows is `/winnt/krb5.ini` and on a distributed environment is `/etc/krb5`. If you specify another location path, then you must also specify the java.security.krb5.conf JVM property.

For example, if your `krb5.conf` file is specified at `/opt/IBM/WebSphere/profiles/AppServer/etc/krb5.conf`, then you need to specify `-Djava.security.krb5.con=/opt/IBM/WebSphere/profiles/AppServer/etc/krb5.conf`.

Creating the Kerberos keytab file:

You use the **ktpass** tool to create the Kerberos keytab file (`krb5.keytab`) for use with the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Below is a summary of the functions available when you enter `ktpass -?` on the command line.

```
C:\MS SDK>ktpass -?
Command line options:

---------------------most useful args
[- /]          out : Keytab to produce
[- /]        princ : Principal name (user@REALM)
[- /]         pass : password to use
                     use "*" to prompt for password.
---------------------less useful stuff
[- /]      mapuser : map princ (above) to this user account (default: don't)
[- /]        mapOp : how to set the mapping attribute (default: add it)
[- /]        mapOp :  is one of:
[- /]        mapOp :        add : add value (default)
[- /]        mapOp :        set : set value
[- +]      DesOnly : Set account for des-only encryption (default:no)
[- /]           in : Keytab to read/digest
---------------------options for key generation
[- /]       crypto : Cryptosystem to use
[- /]       crypto :  is one of:
[- /]       crypto : DES-CBC-CRC : for compatibility
[- /]       crypto : DES-CBC-MD5 : default
[- /]       crypto :        RC4 :
[- /]        ptype : principal type in question
[- /]        ptype :  is one of:
[- /]        ptype : KRB5_NT_PRINCIPAL : The general ptype-- recommended
[- /]        ptype : KRB5_NT_SRV_INST : user service instance
[- /]        ptype : KRB5_NT_SRV_HST : host service instance
[- /]         kvno : Override Key Version Number
                     Default: query DC for kvno.  Use /kvno 1 for Win2K compat.
[- +]       Answer : +Answer answers YES to prompts.  -Answer answers NO.
[- /]       Target : Which DC to use.  Default:detect
```

You use the **ktpass** tool in the following ways to create the Kerberos keytab file:

For a single DES encryption type

From a command prompt, run the **ktpass** command:

```
ktpass -out c:\temp\server3.keytab
-princ HTTP/server3.wasteched30.torolab.ibm.com@WASTECHED30.TOROLAB.IBM.COM
-mapUser server3
-mapOp set -pass was1edu
-crypto DES-CBC-MD5
+DesOnly
```

Table 11. Using ktpass for a single DES encryption type

Option	Explanation
`-out c:\temp\server3.keytab`	The key is written to this output file.
`-princ HTTP/server3. wasteched30.torolab.ibm.com @WASTECHED30.TOROLAB.IBM.COM`	The concatenation of the user logon name, and the realm must be in uppercase.
`-mapUser`	The key is mapped to the user, server3.
`-mapOp`	This option sets the mapping.
`-pass was1edu`	This option is the password for the user ID.
`-crypto DES-CBC-MD5`	This option uses the single DES encryption type.
`+DesOnly`	This option generates only DES encryptions.

For the RC4-HMAC encryption type

Important: RC4-HMAC encryption is only supported when using a Windows 2003 Server as KDC. RC4-HMAC encryption is not supported with a Windows 2000 Server as KDC.

From a command prompt, run the **ktpass** command.

```
ktpass -out c:\temp\poc.keytab
-princ HTTP/interop.wasteched2.torolab.ibm.com@WASTECHED2.TOROLAB.IBM.COM
-mapUser server3
-mapOp set
–pass was1edu
-crypto RC4
```

Table 12. Using ktpass for the RC4-HMAC encryption type

Option	Explanation
`-out f:\abc\server3.keytab`	The key is written to this output file.
`-princ HTTP/server3. wasteched30.torolab.ibm.com @WASTECHED30.TOROLAB.IBM.COM`	The concatenation of the user logon name, and the realm must be in uppercase.
`-mapUser`	The key is mapped to the user, server3.
`-mapOp`	This option sets the mapping.
`-pass was1edu`	This option is the password for the user ID.
`-crypto RC4`	This option chooses the RC4-HMAC encryption type.

The Kerberos keytab file is created for use with the SPNEGO TAI.

Configuring SPNEGO TAI in WebSphere Application Server:

Performing this task helps you, as Web administrator, to ensure that WebSphere Application Server is properly configured to enable the operation of the Simple and Protected GSS-API Negotiation (SPNEGO) trust association interceptor (TAI).

You need to know how to use the WebSphere Application Server administrative console to manage the security configuration and have the proper authority to modify the security configuration of the application server.

Note: It is recommended that you use wsadmin to manage the SPNEGO TAI properties.

Complete the following steps to enable the operation of the SPNEGO TAI.

1. Log on to the WebSphere Application Server administrative console.
2. Click **Security > Secure administration, applications, and infrastructure**.
3. Expand **Web security** and click **Trust association**.
4. Under the General Properties heading, select the **Enable trust association** check box, then click **Interceptors**.
5. Select the SPNEGO TAI in the list of interceptors, then click **Custom properties**.
6. Click **New** and then fill in the **Name** and **Value** fields. Click **OK**. Repeat this step for each custom property that you want to apply to the SPNEGO TAI.
7. After you finish defining your custom properties, click **Save** to store the updated SPNEGO TAI configuration.

Your SPNEGO TAI configuration is now configured for WebSphere Application Server. You must ensure that:

- A user account is created in the Microsoft Active Directory and mapped to a Kerberos principal name.
- A Kerberos keytab file (`krb5.keytab`) is created and made available to the WebSphere Application Server. The Kerberos keytab file contains keys WebSphere Application Server uses to authenticate the user in the Microsoft Active Directory and the Kerberos account.

SPNEGO TAI custom configuration attributes:

The Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) custom configuration attributes control different operational aspects of the SPNEGO TAI. You can specify different attribute values for each application server.

Each of the attributes defined in the following table is specified in the Custom Properties panel for the SPNEGO TAI using the administrative console facility. For convenience, you can optionally place these attributes in a properties file. In this case, the SPNEGO TAI loads the configuration attributes from the file instead of the Custom Properties panel definition. Refer to com.ibm.ws.security.spnego.propertyReloadFile property as defined in “SPNEGO TAI JVM configuration attributes” on page 1264.

To assign unique attribute names that identify each possible SPN, an SPN<id> is embedded in the attribute name and used to group the attributes that are associated with each SPN. The SPN<id> s are numbered sequentially for each attribute group.

Table 13.

Attribute Name	Required	Default Value
com.ibm.ws.security.spnego.SPN<id>.hostName	Yes	None
com.ibm.ws.security.spnego.SPN<id>.filterClass	No	See the description that follows.
com.ibm.ws.security.spnego.SPN<id>.filter	No	See the description that follows.
com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate	No	false
com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage	No	See the description that follows.
com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage	No	See the description that follows.
com.ibm.ws.security.spnego.SPN<id>.trimUserName	No	true

com.ibm.ws.security.spnego.SPN<id>.hostName

This attribute is required. It specifies the hostname in the SPN used by the SPNEGO TAI to establish a Kerberos secure context.

Note: The hostname is the long form of hostname. For example, `myHostName.austin.ibm.com`. The Kerberos SPN is a string of the form `HTTP/hostname@realm`. The complete SPN is used with the Java Generic Security Service (JGSS) by the SPNEGO provider to obtain the security credential and security context that are used in the authentication process.

com.ibm.ws.security.spnego.SPN<id>.filterClass

This attribute is optional. It specifies the name of the Java class that is used by the SPNEGO TAI to select which HTTP requests are subject to SPNEGO authentication. If no class is specified, the default com.ibm.ws.security.spnego.HTTPHeaderFilter implementation class is used. The Java class that is specified must implement the `com.ibm.wsspi.security.spnego.SpnegoFilter` interface. A default implementation of this interface is provided. Specify the `com.ibm.ws.security.spnego.HTTPHeaderFilter` class to use the default implementation. This class uses the selection rules specified with the `com.ibm.ws.security.spnego.SPN<id>.filter` property.

com.ibm.ws.security.spnego.SPN<id>.filter

This attribute is optional. It defines the filtering criteria that is used by the specified class with the previous attribute. It defines arbitrary criteria that is meaningful to the implementation class used. The com.ibm.ws.security.spnego.HTTPHeaderFilter default implementation class uses this attribute to define a list of selection rules that represent conditions that are matched against the HTTP request headers to determine whether or not the HTTP request is selected for SPNEGO authentication.

Each condition is specified with a key-value pair, separated from each other by a semicolon. The conditions are evaluated from left to right, as they display in the specified attribute. If all conditions are met, the HTTP request is selected for SPNEGO authentication.

The key and value in the key-value pair are separated by an operator that defines which condition is checked. The key identifies an HTTP request header to extract from the request and its value is compared with the value that is specified in the key-value pair according to the operator specification. If the header that is identified by the key is not present in the HTTP request, the condition is treated as not being met.

Any of the standard HTTP request headers can be used as the key in the key-value pairs. Refer to the HTTP specification for the list of valid headers. In addition, two keys are defined to extract information from the request, also useful as a selection criterion, which is not available through standard HTTP request headers. The remote-address key is used as a pseudo header to retrieve the remote TCP/IP address of the client application that sent the HTTP request. The request-URL key is used as a pseudo header to retrieve the URL that is used by the client application to make the request. The interceptor uses the result of the getRequestURL operation in the javax.servlet.http.HttpServletRequest interface to construct the Web address. If a query string is present, the result of the getQueryString operation in the same interface is also used. In this case, the complete URL is constructed as follows:

```
String url = request.getRequestURL() + '?' + request.getQueryString();
```

The following operators and conditions are defined:

Table 14. Filter conditions and operations

Condition	Operator	Example
Match exactly	= = Arguments are compared as equal.	host=host.my.company.com
Match partially (includes)	%= Arguments are compared with a partial match being valid.	user-agent%=IE 6

Table 14. Filter conditions and operations (continued)

Condition	Operator	Example
Match partially (includes one of many)	^= Arguments are compared with a partial match being valid for one of many arguments specified.	request-url^=webApp1|webApp2|webApp3
Does not match	!= Arguments are compared as not equal.	request-url!=noSPNEGO
Greater than	> Arguments are compared lexogaphically as greater than.	remote-address>192.168.255.130
Less than	< Arguments are compared lexographically as less than.	remote-address<192.168.255.135

com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate
This attribute is optional. It indicates whether or not the Kerberos SPNEGO delegated credentials are stored by the SPNEGO TAI. This attribute enables the capability for an application to retrieve the stored credentials and propagate them to other applications downstream for additional SPNEGO authentication.

This attribute requires use of the advanced Kerberos credential delegation feature and requires development of custom logic by the application developer. The developer must interact directly with the Kerberos Ticket Granting Service (TGS) to obtain a Ticket Granting Ticket (TGT) using the delegated Kerberos credentials on behalf of the end-user who originated the request. The developer must also construct the appropriate Kerberos SPNEGO token and include it in the HTTP request to continue the downstream SPNEGO authentication process, including handling additional SPNEGO challenge-response exchange, if necessary.

com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage
This attribute is optional. It specifies the Web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays if it does not support SPNEGO authentication. It can specify a Web (http://) or a file (file://) resource. If this attribute is not specified or the interceptor cannot find the specified resource, the following content is used:

```
<html><head><title>SPNEGO authentication is not supported</title></head>
<body>SPNEGO authentication is not supported on this client</body></html>;
```

com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage
This attribute is optional. It specifies the Web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays when the SPNEGO token is received by the interceptor when the challenge-response handshake contains a NT LAN Manager (NTLM) token instead of the expected SPNEGO token. It can specify a Web (http://) or a file (file://) resource. If this attribute is not specified or the interceptor cannot find the specified resource, the following content is used:

```
<html><head><title>An NTLM Token was received.</title></head>
<body>Your browser configuration is correct, but you have not logged into a supported
Microsoft(R) Windows(R) Domain.
<p>Please login to the application using the normal login page.</html>
```

com.ibm.ws.security.spnego.SPN<id>.trimUserName
This attribute is optional. It specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the ″@″ that precedes the Kerberos realm name. If this

attribute is set to `true`, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`. For example,

When com.ibm.ws.security.spnego.SPN<id>.trimUserName = true

```
bobsmith@myKerberosRealm  becomes  bobsmith
```

When com.ibm.ws.security.spnego.SPN<id>.trimUserName = false

```
bobsmith@myKerberosRealm  remains  bobsmith@myKerberosRealm
```

Note: The following commands tasks can be used to operate on these SPNEGO TAI attributes:

- "Adding SPNEGO TAI properties using the wsadmin utility"
- "Deleting SPNEGO TAI properties using the wsadmin utility" on page 1261
- "Modifying SPNEGO TAI properties using the wsadmin utility" on page 1259
- "Displaying SPNEGO TAI properties using the wsadmin utility" on page 1262

Related concepts

"Single sign-on for HTTP requests using SPNEGO" on page 1199
WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server.

Related tasks

"Adding SPNEGO TAI properties using the wsadmin utility"
You use the wsadmin utility to add properties for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) in the security configuration for WebSphere Application Server.

"Deleting SPNEGO TAI properties using the wsadmin utility" on page 1261
You use the wsadmin utility to delete properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

"Modifying SPNEGO TAI properties using the wsadmin utility" on page 1259
You use the wsadmin utility to modify the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

"Displaying SPNEGO TAI properties using the wsadmin utility" on page 1262
You use the wsadmin utility to display the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

"Mapping user Ids from client to server for SPNEGO" on page 1267
You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by implementing arbitrary mappings of the end-user's identity, which is retrieved from Microsoft Active Directory to the identity that is used in the WebSphere Application Server security registry.

Adding SPNEGO TAI properties using the wsadmin utility:

You use the wsadmin utility to add properties for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) in the security configuration for WebSphere Application Server.

Use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.

3. At the **wsadmin** prompt, enter the following command:

```
$AdminTask addSpnegoTAIProperties
```

You can use the following parameters with this command:

Option	Description
<spnId>	This is the SPN identifier for the group of custom properties that are to be defined with this command. If you do not specify this parameter, an unused SPN identifier is assigned.
<host>	It specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. This parameter is required.
<filter>	This attribute is optional. It defines the filtering criteria used by the class specified with the above attribute. If no filter is specified, all HTTP requests are subject to SPNEGO authentication.
<filterClass>	This attribute is optional. It specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no filter class is specified, the default filter class, com.ibm.ws.security.spnego.HTTPHeaderFilter, is used.
<noSpnegoPage>	This attribute is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. If you do not specify the noSpnegoPage attribute then the default is used: `"<html><head><title>SPNEGO authentication is not supported. </title></head>" + "<body>SPNEGO authentication is not supported on this client. </body></html>";`
<ntlmTokenPage>	This attribute is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. If you do not specify the ntlmTokenPage attribute then the default is used: `"<html><head><title>An NTLM Token was received.</title></head>" + "<body>Your browser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";`

Option	Description
<trimUserName>	This parameter is optional. It specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the ″@″ that precedes the Kerberos realm name. If this attribute is set to `true`, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`.

SPNEGO TAI properties have been added for this WebSphere Application Server.

Example 1

The following example configures the SPNEGO TAI to intercept HTTP request that contain `IE 6` in the user agent request header. The SPNEGO TAI uses the SPN of HTTP/ myhost.ibm.com@<default_realm> to authenticate the request originator.

```
$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6
```

Example 2

The following is an example of adding SPNEGOTAIProperties for SPN1 to use the default filterClass and to intercept all requests for the host, central01.austin.ibm.com.

```
wsadmin>$AdminTask addSpnegoTAIProperties -interactive
Add SPNEGO TAI properties

Add SPNEGO TAI configuration properties.

*Host name in Service Principal Name (host): central01.austin.ibm.com
Service Principal Name identifier (spnId): 1
HTTP header filter rule (filter):
Name of class used to filter HTTP requests (filterClass):
SPNEGO not supported browser response (noSpnegoPage):
NTLM Token received browser response (ntlmTokenPage):
Trim User Name browser response (trimUserName):

Add SPNEGO TAI properties

F (Finish)
C (Cancel)

Select [F, C]: [F] f
WASX7278I: Generated command line: $AdminTask addSpnegoTAIProperties {-host central01.austin.ibm.com}
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
wsadmin>
```

Modifying SPNEGO TAI properties using the wsadmin utility:

You use the wsadmin utility to modify the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.
3. At the **wsadmin** prompt, enter the following command:

   ```
   $AdminTask modifySpnegoTAIProperties
   ```

You can use the following parameters with this command:

Option	Description
<spnId>	The SPN identifier for the group of custom properties that are to be defined with this command. You must specify this parameter.
<host>	This parameter is optional. It specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context.
<filter>	This parameter is optional. It defines the filtering criteria used by the class specified with the above attribute.
<filterClass>	This parameter is optional. It specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no class is specified, all HTTP requests will be subject to SPNEGO authentication.
<noSpnegoPage>	This parameter is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. If you do not specify the noSpnegoPage attribute then the default is used: `"<html><head><title>SPNEGO authentication is not supported. </title></head>" + "<body>SPNEGO authentication is not supported on this client. </body></html>";`
<ntlmTokenPage>	This parameter is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. If you do not specify the ntlmTokenPage attribute then the default is used: `"<html><head><title>An NTLM Token was received.</title></head>" + "<body>Your browser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";`
<trimUserName>	This parameter is optional. It specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the ″@″ that precedes the Kerberos realm name. If this attribute is set to `true`, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`.

SPNEGO TAI properties are modified for this WebSphere Application Server.

Example 1

The following example configures the SPNEGO TAI to intercept HTTP request that contain `IE 6` in the user agent request header. The SPNEGO TAI uses the SPN of HTTP/ myhost.ibm.com@<default_realm> to authenticate the request originator. Then the example modifies the value of the filter custom property that was defined and changes it from `user-agent%=IE 6` to `host==myhost.company.com`.

```
$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6
$AdminTask modifySpnegoTAIProperties -spnId 1 -filter host==myhost.company.com
```

Example 2

This is an example of modifying the SPNEGO TAI for SPN1 properties to add a filter for host central01.austin.ibm.com.

```
wsadmin>$AdminTask modifySpnegoTAIProperties -interactive
Modify SPNEGO TAI properties

Modify SPNEGO TAI configuration properties

*Service Principal Name identifier (spnId): 1
Host name in Service Principal Name (host): central01.austin.ibm.com
HTTP header filter rule (filter): request-url!=noSPNEGO;request-url%=snoop
Name of class used to filter HTTP requests (filterClass):
SPNEGO not supported browser response (noSpnegoPage):
NTLM Token received browser response (ntlmTokenPage):
Trim User Name browser response (trimUserName):

Modify SPNEGO TAI properties

F (Finish)
C (Cancel)

Select [F, C]: [F] f
WASX7278I: Generated command line: $AdminTask modifySpnegoTAIProperties {-spnId
1 -host w2003secdev.austin.ibm.com -filter request-url!=noSPNEGO;request-url%=sn
oop}
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
wsadmin>
```

Deleting SPNEGO TAI properties using the wsadmin utility:

You use the wsadmin utility to delete properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.
3. At the **wsadmin** prompt, enter the following command:

   ```
   $AdminTask deleteSpnegoTAIProperties
   ```

You can use the following parameters with this command:

Option	Description
<spnId>	This is an optional parameter. The SPN identifier for the group of custom properties that are to be deleted with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are deleted.

SPNEGO TAI properties are deleted for this WebSphere Application Server.

Example 1

The following example deletes all the SPNEGO TAI properties for SPN2

```
wsadmin>$AdminTask deleteSpnegoTAIProperties {-spnId 2}
```

Example 2

The following example deletes all SPNEGO TAI properties

```
wsadmin>$AdminTask deleteSpnegoTAIProperties
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN2.hostName=wssecpd.austin.ibm.com
wsadmin>
```

Displaying SPNEGO TAI properties using the wsadmin utility:

You use the wsadmin utility to display the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root*/bin directory.
3. At the **wsadmin** prompt, enter the following command:

   ```
   $AdminTask showSpnegoTAIProperties
   ```

   You can use the following parameters with this command:

Option	Description
<spnId>	This is an optional parameter. The service principal name (SPN) identifier for the group of custom properties that are to be displayed with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are displayed.

SPNEGO TAI properties are displayed for this WebSphere Application Server.

Example 1

The following example displays all SPNEGO TAI properties.

```
wsadmin>$AdminTask showSpnegoTAIProperties
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN2.hostName=wssecpd.austin.ibm.com
wsadmin>
```

Example 2

The following example displays SPNEGO TAI properties for SPN1 and host, central01.austin.ibm.com.

```
wsadmin>$AdminTask showSpnegoTAIProperties -interactive
Show SPNEGO TAI configuration properties.

Display SPNEGO TAI configuration properties.

Service Principal Name identifier (spnId): 1

Show SPNEGO TAI configuration properties.

F (Finish)
C (Cancel)

Select [F, C]: [F]
WASX7278I: Generated command line: $AdminTask showSpnegoTAIProperties {-spnId 1}

com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN1.trimUserName=true
wsadmin>
```

Configuring JVM properties and enabling SPNEGO TAI in WebSphere Application Server:

Performing this task helps you, as Web administrator, to ensure that WebSphere Application Server is configured to enable the operation of the Simple and Protected GSS-API Negotiation mechanism (SPNEGO) trust association interceptor (TAI) with the required Java virtual machine (JVM) property.

You need to know how to use the WebSphere Application Server administrative console to manage the security configuration and have the proper authority to modify the security configuration of the application server.

Complete the following steps to enable the operation of the SPNEGO TAI by setting the JVM required property.

1. Log on to WebSphere Application Server administrative console.
2. Click **Servers > Application servers**.
3. Select the appropriate servers and click **Java and process management > Process Definition**.
4. Click **Java virtual machine**. In the **Generic JVM arguments** field, type `-Dcom.ibm.ws.security.spnego.isEnabled=true`.
5. Click **Apply > OK** to save the configuration

The application server is configured and ready to provide a single sign-on environment for end users who have successfully authenticated in a Microsoft Active Directory domain. You must restart each application server that is configured for SPNEGO Web authentication.

Enabling the SPNEGO TAI using scripting:

You use the wsadmin utility to enable the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to enable the SPNEGO TAI:

1. Identify the server and assign it to the `server1` variable:
   - Using Jacl:

     ```
     set server1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```

- Using Jython:

```
server1 = AdminConfig.getid("Cell:mycell/Node:mynode/Server:server1/")
print server1
```

Example output:

```
server1(cells/mycell/nodes/mynode|servers/seerver1|server.xml#Server_1)
```

2. Identify the Java virtual machine (JVM) belonging to this server and assign it to the `jvm` variable:
   - Using Jacl:

```
set jvm [$AdminConfig list JavaVirutalMachine $server1]
```

   - Using Jython:

```
jvm = AdminConfig.list('JavaVirutalMachine',server1')
```

   Example output:

```
(cells/mycell/nodes/mynode/servers/server1:server.xml#JavaVirtualMachine_1)
(cells/mycell/nodes/mynode/servers/server1:server.xml#JavaVirtualMachine_2)
```

3. Identify the controller JVM of the server:
   - Using Jacl:

```
set cjvm [lindex $jvm 0]
```

   Using Jython:

```
# get line separator
import java
lineSeparator = java.lang.System.getProperty('line.separator')
arrayJVMs = jvm.split(lineSeparator)
cjvm = arrayJVMs[0]
```

4. Modify the generic JVM arguments to enable SPNEGO TAI:
   - Using Jacl:

```
$AdminConfig modify $cjvm { {genericJvmArguments "-Dcom.ibm.ws.security.spnego.isEnabled=true"} }
```

   - Using Jython:

```
AdminConfig.modify(cjvm, [['genericJvmArguments', "-Dcom.ibm.ws.security.spnego.isEnabled=true"]])
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
6. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

SPNEGO TAI JVM configuration attributes:

Java virtual machine (JVM) attributes control the operation of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI).

The following JVM attributes control operation of the SPNEGO TAI. Different attribute values can be specified for each application server.

Table 15. JVM configuration attributes

Attribute Name	Required	Value Type	Default Value	Recommended Value
com.ibm.ws.security.spnego.isEnabled	No	Boolean	False	True
com.ibm.ws.security.spnego.propertyReloadFile	No	String	None	For Windows `C:\temp\TAI.props` For UNIX `/tmp/TestTAI.Properties`
com.ibm.ws.security.spnego.propertyReloadTimeout	No	Integer	None	120

com.ibm.ws.security.spnego.isEnabled
Use this attribute to enable or disable operation of the SPNEGO TAI in a given application server. When set to `false`, the SPNEGO TAI is disabled and not used by the Web authentication module for authenticating any Web requests. When set to `true`, the SPNEGO TAI is enabled and used by the Web authentication module for authenticating any Web requests.

com.ibm.ws.security.spnego.propertyReloadFile
Use this attribute to identify the file that contains configuration properties for the SPNEGO TAI, when it is not convenient to stop and restart the application server. The properties contained in this file can be reloaded to configure the SPNEGO TAI.

Important: the properties that are defined in the specified file override any properties defined using the administrative console.

A sample of this reload file follows:

```
#########################################################
# Template properties files for SPNEGO TAI
#
# Where possible defaults have been provided.
#
#########################################################

#---------------------------------------------------------
# Hostname
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.HostName=wsecurity.austin.ibm.com

#---------------------------------------------------------
# (Optional) SpnegoNotSupportedPage
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.SpnegoNotSupportedPage=

#---------------------------------------------------------
# (Optional) NTLMTokenReceivedPage
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.NTLMTokenReceivedPage=

#---------------------------------------------------------
# (Optional) FilterClass
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.FilterClass=com.ibm.ws.spnego.HTTPHeaderFilter

#---------------------------------------------------------
# (Optional) Filter
#---------------------------------------------------------
#com.ibm.ws.spnego.SPN1.Filter=
```

Important: If com.ibm.ws.security.spnego.propertyReloadFile attribute is set, but the com.ibm.ws.security.spnego.propertyReloadTimeout attribute is not, then the SPNEGO TAI is not initialized.

com.ibm.ws.security.spnego.propertyReloadTimeout
Use this attribute to specify a time interval in seconds that elapses after which the SPNEGO TAI reloads the configuration properties. Also, the SPNEGO TAI reloads the configuration properties if the file that is identified by the com.ibm.ws.security.spnego.propertyReloadFile attribute changed since the last time the configuration attributes were retrieved. This time interval in seconds must be specified as a positive integer.

Important:

- If the com.ibm.ws.security.spnego.propertyReloadFile attribute and the com.ibm.ws.security.spnego.propertyReloadTimeout attribute are not set, then the SPNEGO

TAI properties are only loaded once from the SPNEGO TAI custom properties defined in the WebSphere Application Server configuration data. This one time loading occurs when the JVM is initialized.

- If com.ibm.ws.security.spnego.propertyReloadTimeout attribute is set, but the com.ibm.ws.security.spnego.propertyReloadFile attribute is not, then the SPNEGO TAI is not initialized.

The following examples show how to enable operation of the SPNEGO TAI by setting the com.ibm.ws.security.spnego.isEnabled JVM property to `true` using the scripting that is available in WebSphere Application Server for AdminConfig commands.

Using JACL:

```
set server [$AdminConfig getid /Cell:mycell/Node:mynode/Server:myserver]
set jvm [$AdminConfig list JavaVirtualMachine $server]
$AdminConfig modify $jvm {{genericJvmArguments "-Dcom.ibm.ws.security.spnego.isEnabled =true"}}
$AdminConfig save
```

Using Jython:

```
server = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:myserver')
 jvm = AdminConfig.list('JavaVirtualMachine', server)
 AdminConfig.modify(jvm, [['genericJvmArguments', "Dcom.ibm.ws.security.spnego.isEnabled =true"]])
AdminConfig.save()
```

Remember: You can also use the wsadmin command for the AdminConfig scripting object to interactively set the com.ibm.ws.security.spnego.isEnabled attribute. See “Enabling the SPNEGO TAI using scripting” on page 1263 for more information.

The following attributes are not used directly by the SPNEGO TAI; however, they affect the operation of the core security runtime and can also be used for problem determination.

Table 16. JVM configuration attributes

Attribute Name	Required	Value Type	Default Value	Recommended Value
java.security.properties	No	String	None	
com.ibm.security.jgss.debug	No	String	None	″off″ or ″all″
com.ibm.security.krb5.Krb5Debug	No	String	None	″off″ or ″all″
javax.security.auth.useSubjectCredsOnly	Yes	Boolean	True	False

java.security.properties
This property is optional. It can be used when different application servers in a cell have different security requirements and it is not convenient to modify the global `java.security` file for the entire cell. In such situations, the java.security.properties attribute is used to specify the location of the `java.security` file used by the JVM for each application server.

com.ibm.security.jgss.debug
This attribute is optional. It can be used to collect diagnostic trace information for problem determination in the Java Generic Security Service (JGSS) application programmer interface (API) implementation. The value can be set to `all` or `off` to enable or disable tracing, respectively. See Java Generic Security Service User’s Guide for specific JGSS API information.

com.ibm.security.krb5.Krb5Debug
This attribute is optional. It can be used to collect additional diagnostic trace information for problem determination in the JGSS implementation. The value can be set to `all` or `off` to enable or disable tracing, respectively.

javax.security.auth.useSubjectCredsOnly
JGSS includes an optional Java Authentication and Authorization Service (JAAS) login facility that saves `Principal` credentials and secret keys in the `Subject` of the application’s JAAS login context.

JGSS retrieves credentials and secret keys from the `Subject` by default. This feature can be disabled by setting the Java property javax.security.auth.useSubjectCredsOnly to `false`.

Attention: The SPNEGO TAI does not use the optional JAAS login module. The javax.security.auth.useSubjectCredsOnly property must be set to `false`.

Mapping user Ids from client to server for SPNEGO:

You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by implementing arbitrary mappings of the end-user's identity, which is retrieved from Microsoft Active Directory to the identity that is used in the WebSphere Application Server security registry.

You need to perform some administrative tasks in the WebSphere Application Server environment to use SPNEGO TAI and to ensure that the requester's identity matches the identity in the WebSphere Application Server user registry.

In the simplest deployment of the SPNEGO TAI, it is assumed that the requester's identity in the WebSphere Application Server user registry is identical to the identity retrieved. This is the case when Microsoft Windows Active Directory server is the lightweight directory access protocol (LDAP) server used in WebSphere Application Server. This is default behavior of the SPNEGO TAI.

You do not need to use this simple deployment of the SPNEGO TAI. WebSphere Application Server can use a different registry, such as a local OS, LDAP, or custom registry instead of the Microsoft Active Directory. If WebSphere Application Server uses a different registry than the Microsoft Active Directory, then a mapping from the Microsoft Windows user Id to a WebSphere Application Server user Id is necessary.

1. Configure the Web browser to use SPNEGO.
2. Configure Java virtual machine (JVM) properties and custom SPNEGO TAI properties.
3. Enable the SPNEGO TAI.
4. Use the custom login module to perform any custom mapping of user Ids from the user registry to the user registry of WebSphere Application Server. The custom login module is a plug-in mechanism that is defined for authenticating incoming and outgoing requests in WebSphere Application Server. The custom login module is inserted before the ltpaLoginModule and maps the name in the com.ibm.wsspi.security.tai.TAIResult (which was returned to the Web authenticator) to the corresponding name in the user registry. The ltpaLoginModule then uses the mapped identity to create a WSCredential.

   The custom login module can also supply the full set of security attributes in the javax.security.auth.Subject in the com.ibm.wsspi.security.tai.TAIResult to fully assert the mapped identity. When the identity is fully asserted, the wsMapDefaultInboundLoginModule maps those security attributes to a WSCredential.

Using the custom login module, Microsoft Active Directory identities are mapped to the WebSphere Application Server's security registry.

Filtering HTTP requests for SPNEGO TAI:

You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by specifying whether or not a particular HTTP request should be intercepted.

Before you begin, you need to fully understand the deployment of the SPNEGO TAI in your installation.

Verify the configuration of your SPNEGO TAI. The deployment of the SPNEGO TAI can vary from a single WebSphere Application Server system on which a single application is running to a large multinode

WebSphere Application Server Network Deployment (ND) cell, with dozens of application servers, hosting many applications. Every SPNEGO TAI is installed at the cell level. You must be aware of your particular SPNEGO TAI configuration.

The default behavior of the SPNEGO TAI is to not intercept HTTP requests. This default behavior ensures that the SPNEGO TAI can be installed into an existing cell, configured for a single application server and not change any other application servers in the cell. Other WebSphere Application Servers can run exactly as before within a given configuration.

Then decide whether or not to use the sample SPN<id>.filter class and determine the exact filter properties to use.

Note: The default behavior of the SPNEGO TAI is to use the com.ibm.ws.security.spnego.SPN<id>.filter class and intercept all requests.

If the default behavior is not appropriate, you can use a customer provided class, or extend or modify the sample class as required. The system programmer interface, com.ibm.ws.security.spnego.SpnegoFilter allows you to implement a custom filter to determine whether or not to intercept a particular HTTP request. With the default implementation, you can set filter rules for coarse as well as fine-grained criteria in selecting which HTTP requests to intercept.

1. Set the com.ibm.ws.security.spnego.isEnabled Java virtual machine (JVM) custom property to `true` to enable the SPNEGO TAI on any JVM.
2. Identify when the SPNEGO TAI intercepts a given request. A set of filter properties is provided, but you must determine what is appropriate and modify the com.ibm.ws.security.spnego.SPN<id>.filter class accordingly.

Your SPNEGO TAI is set to filter HTTP requests when it is operating.

Configuring the Web browser to use SPNEGO:

You can configure your browser to utilize the Simple and Protected GSS-API Negotiation (SPNEGO) mechanism. Authentication of your browser requests are processed by the SPNEGO trust association interceptor (TAI) in the WebSphere Application Server.

You need to know how to display and set options in the Microsoft Internet Explorer browser or any other browser (such as Firefox). You must have a browser installed that supports SPNEGO authentication.

Complete the following steps to ensure that your Microsoft Internet Explorer browser is enabled to perform SPNEGO authentication.

1. At the desktop, log in to the windows active directory domain.
2. Activate Internet Explorer.
3. In the Internet Explorer window, click **Tools > Internet Options > Security** tab.
4. Select the **Local intranet** icon and click **Sites**.
5. In the Local intranet window, ensure that the ″check box″ to include all local (intranet) not listed in other zones is selected, then click **Advanced** .
6. In the **Local intranet** window, fill in the Add this Web site to the zone field with the Web address of the host name so that the single sign-on (SSO) can be enabled to the list Web sites shown in the Web sites field. Your site information technology staff provides this information. Click **OK** to complete this step and close the Local intranet window.
7. On the **Internet Options** window, click the **Advanced** tab and scroll to **Security settings**. Ensure that the **Enable Integrated Windows Authentication (requires restart)** box is selected.
8. Click **OK**. Restart your Microsoft Internet Explorer to activate this configuration.

Complete the following steps to ensure that your Firefox browser is enabled to perform SPNEGO authentication.

1. At the desktop, log in to the windows active directory domain.
2. Activate Firefox.
3. At the address field, type **about:config**.
4. In the Filter, type **network.n**
5. If the deployed SPNEGO solution is using the advanced Kerberos feature of Credential Delegation double click on **network.negotiate-auth.delegation-uris**. This preference lists the sites for which the browser may delegate user authorization to the server.
6. Enter a comma delimited list of trusted domains or URLs.
7. Click **OK**. The configuration appears as updated.
8. Restart your Firefox browser to activate this configuration.

Your Internet browser is properly configured for SPNEGO authentication. You can use applications that are deployed in WebSphere Application Server that use secured resources without being repeatedly requested for an ID and password.

Configuring single sign-on capability with Tivoli Access Manager or WebSEAL

Either Tivoli Access Manager WebSEAL or Tivoli Access Manager plug-in for Web servers can be used as reverse proxy servers to provide access management and single sign-on (SSO) capability to WebSphere Application Server resources. With such an architecture, either WebSEAL or the plug-in authenticates users and forwards the collected credentials to WebSphere Application Server in the form of an IV Header. Two types of single sign-on are available, the TAI interface and the TAI++ interface, so named as both use WebSphere Application Server trust association interceptors (TAI). With the TAI, the end-user name is extracted from the HTTP header and forwarded to embedded Tivoli Access Manager where the end-user name is used to construct the client credential information and authorize the user. With the TAI++, all of the user credential information is available in the HTTP header and not just the user name. The TAI++ is the more efficient of the two solutions because a Lightweight Directory Access Protocol (LDAP) call is not required. TAI functionality is retained for backwards compatibility.

Complete the following tasks to enable single sign-on to WebSphere Application Server using either WebSEAL or the plug-in for Web servers. These tasks assume that embedded Tivoli Access Manager is configured for use.

1. Create a trusted user account for Tivoli Access Manager in the shared Lightweight Directory Access Protocol (LDAP) user registry. For more information, see “Creating a trusted user account in Tivoli Access Manager” on page 1275.
2. Configure either WebSEAL or the Tivoli Access Manager plug-in for Web servers to work with WebSphere Application Server. For more information, see either of the following articles:
   - “Configuring WebSEAL for use with WebSphere Application Server” on page 1275
   - “Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server” on page 1276
3. Configure single sign-on using either the TAI or TAI++ interface. For more information, see either of the following articles:
   - “Configuring single sign-on using the trust association interceptor” on page 1277
   - “Configuring single sign-on using trust association interceptor ++” on page 1278

Single sign-on settings:

Use this page to set the configuration values for single sign-on (SSO).

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Web security > Single sign-on (SSO)**.

Enabled:

Specifies that the single sign-on function is enabled.

Web applications that use J2EE FormLogin style login pages, such as the administrative console, require single sign-on (SSO) enablement. Only disable SSO for certain advanced configurations where LTPA SSO-type cookies are not required.

Data type:	Boolean
Default:	Enabled
Range:	Enabled or Disabled

Requires SSL:

Specifies that the single sign-on function is enabled only when requests are made over HTTPS Secure Sockets Layer (SSL) connections.

Data type:	Boolean
Default:	Disable
Range:	Enable or Disable

Domain name:

Specifies the domain name (.ibm.com, for example) for all single sign-on hosts.

The application server uses all the information after the first period, from left to right, for the domain names. If this field is not defined, the Web browser defaults the domain name to the host name where the Web application is running. Also, single sign-on is then restricted to the application server host name and does not work with other application server host names in the domain.

You can specify multiple domains separated by a semicolon (;), a space ( ), a comma (,), or a pipe (|). Each domain is compared with the host name of the HTTP request until the first match is located. For example, if you specify ibm.com;austin.ibm.com and a match is found in the ibm.com domain first, the application server does not match the austin.ibm.com domain. However, if a match is not found in either ibm.com or austin.ibm.com, then the application server does not set a domain for the LtpaToken cookie.

If you specify the `UseDomainFromURL` value, the application server sets the SSO domain name value to the domain of the host that is used in the Web address. For example, if an HTTP request comes from `server1.raleigh.ibm.com`, the application server sets the SSO domain name value to `raleigh.ibm.com`.

Tip: The UseDomainFromURL value is case insensitive. You can type `usedomainfromurl` to use this value.

Data type:	String

Interoperability mode:

Specifies that an interoperable cookie is sent to the browser to support back-level servers.

In WebSphere Application Server, Version 6 and later, a new cookie format is needed by the security attribute propagation functionality. When the interoperability mode flag is enabled, the server can send a maximum of two single sign-on (SSO) cookies back to the browser. In some cases, the server just sends the interoperable SSO cookie.

Web inbound security attribute propagation:

When Web inbound security attribution propagation is enabled, security attributes are propagated to front-end application servers. When this option is disabled, the single sign-on (SSO) token is used to log in and recreate the Subject from the user registry.

If the application server is a member of a cluster and the cluster is configured with a distributed replication service (DRS) domain, then propagation occurs. If DRS is not configured, then the SSO token contains the originating server information. With this information, the receiving server can contact the originating server using an MBean call to get the original serialized security attributes.

com.tivoli.pd.jcfg.PDJrteCfg utility for Tivoli Access Manager single sign-on:

The com.tivoli.pd.jcfg.PDJrteCfg utility configures the Java Runtime Environment component for Tivoli Access Manager. This component enables WebSphere Application Server to use Tivoli Access Manager security.

Purpose

Syntax

```
java com.tivoli.pd.jcfg.PDJrteCfg -action {config | unconfig} -cfgfiles_path
configuration_file_path -host policy_server_host jre_path]
```

Parameters

-action {config|unconfig}
Specifies the action to be performed. Actions include:

config Use to configure the Access Manager Java Runtime Environment component.

unconfig
Use to reconfigure the Access Manager Java Runtime Environment component.

-host *policy_server_host*
Specifies the policy server host name.

Valid values for *policy_server_host* include any valid IP host name.

Examples include:

```
host = libra
host = libra.dallas.ibm.com
```

Comments

This command copies Tivoli Access Manager Java libraries to a library extensions directory that exists for a Java runtime that has already been installed on the system.

You can install more than one Java Runtime Environment (JRE) on a given machine. The `pdjrtecfg` command can be used to configure the Tivoli Access Manager Java Runtime Environment component independently for each of the JRE configurations.

```
${JAVA_HOME}/bin/java
-Dfile.encoding=ISO8859-1 \
-Dws.output.encoding=CP1047 \
-Xnoargsconversion \
-Dpd.home=${WAS_HOME}/java/jre/PolicyDirector \
-cp ${WAS_HOME}/java/jre/lib/ext/PD.jar \
 com.tivoli.pd.jcfg.PDJrteCfg \
```

```
-action config \
          -cfgfiles_path ${WAS_HOME}/java/jre \
          -host gary.us.ibm.com \
```

com.tivoli.pd.jcfg.SvrSslCfg utility for Tivoli Access Manager single sign-on:

The utility is used to configure and remove the configuration information associated with WebSphere Application Server and the Tivoli Access Manager server.

Purpose

Syntax

```
java com.tivoli.pd.jcfg.SvrSslCfg
-action {config | unconfig} -admin_id admin_user_ID
-admin_pwd admin_password -appsvr_id application_server_name
-appsvr_pwd application_server_password -mode{local|remote}
-host host_name_of_application_server
-policysvr policy_server_name:port:rank [,...]
-authzsvr authorization_server_name:port:rank [,...]
-cfg_file fully_qualified_name_of_configuration_file
-domain Tivoli_Acccess_Manager_domain
-key_file fully_qualified_name_of_keystore_file
-cfg_action {create|replace}
```

Parameters

-action {config | unconfig}
Specifies the configuration action that is performed by the script. The following options apply:

-action config
Configuring a server creates user and server information in the user registry and creates local configuration and key store files on the application server. Use the -action unconfig option to reverse this operation.

If this action is specified, the following options are required: -admin_id, -admin_pwd, -appsvr_id, -port, -mode, -policysvr, -authzsvr, and -key_file.

-action unconfig
Reconfigures an application server to complete the following actions:

- Remove the user and server information from the user registry
- Delete the local key store file
- Remove information for this application from the configuration file without deleting the file

The reconfiguration operation fails only if the caller is unauthorized or the policy server cannot be contacted.

This action can succeed when a configuration file does not exist. When the configuration file does not exist, it is created and used as a temporary file to hold configuration information during the operation, and then the file is deleted completely.

If this action is specified, the following options are required: -admin_id, -admin_pwd, -appsvr_id, and -policysvr.

-admin_id *admin_user_ID*
Specifies the Tivoli Access Manager administrator name. If this option is not specified, sec_master is the default.

A valid administrative ID is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the administrative ID.

For example, for U.S. English the valid characters are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the administrative ID, if there are limits, are imposed by the underlying registry.

-admin_password *admin_password*

Specifies the password of the Tivoli Access Manager administrator user that is associated with the **-admin_id** parameter. The password restrictions depend upon the password policy for your Tivoli Access Manager configuration.

-appsvr_id *application_server_name*
Specifies the name of the application server. The name is combined with the host name to create unique names for Tivoli Access Manager objects created for your application. The following names are reserved for Tivoli Access Manager applications: ivacld, secmgrd, ivnet, and ivweb.

-appsvr_pwd *application_server_password*
Specifies the password of the application server. This option is required. A password is created by the system and the configuration file is updated with the password created by the system.

If this option is not specified, the server password will be read from standard input.

-authzsvr *authorization_server_name*
Specifies the name of the Tivoli Access Manager authorization server with which the application server communicates. The server is specified by fully qualified host name, the SSL port number, and the rank. The default SSL port number is 7136. For example: myauth.mycompany.com:7136:1. You can specify multiple servers if the entries are separated by a comma (,).

-cfg_action {create | replace}
Specifies the action to take when creating the configuration and key files. Valid values are **create** or **replace**. Use the **create** option to initially create the configuration and keystore files. Use the **replace** option if these files already exist. If you use the **create** option and the configuration or keystore files already exist, an exception is created.

Options are as follows:

create Specifies to create the configuration and key store files during server configuration. Configuration fails if either of these files already exists.

replace
Specifies to replace the configuration and key store files during server configuration. Configuration deletes any existing files and replaces them with new ones.

-cfg_file *fully_qualified_name_of_configuration_file*
Specifies the configuration file path and name.

A file name should be an absolute file name (fully qualified file name) to be valid.

-domain *Tivoli_Access_Manager_domain*
Specifies the Tivoli Access Manager domain name to which the administrator is authenticated. This domain must exist and an the administrator ID and password must be valid for this domain. The application server is specified in this domain.

If not specified, the local domain that was specified during Tivoli Access Manager runtime configuration will be used. The local domain value will be retrieved from the configuration file.

A valid domain name is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the domain name.

For example, for U.S. English the valid characters for domain names are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the domain name, if there are limits, are imposed by the underlying registry.

-host *host_name_of_application_server*
Specifies the TCP host name used by the Tivoli Access Manager policy server to contact this server. This name is saved in the configuration file using the azn-app-host key.

The default is the local host name returned by the operating system. Valid values for host_name include any valid IP host name.

Examples:

host = libra
host = libra.dallas.ibm.com

-key_file *fully_qualified_name_of_keystore_file*
Specifies the directory that is to contain the key files for the server. A valid directory name is determined by the operating system. Use a fully qualified file name that contains the application server certificate and key file.

Make sure that server user (for example, ivmgr) or all users have permission to access the .kdb file and the folder that contains the .kdb file.

This option is required.

-mode *server_mode*
Specifies the mode in which the application operates. This value must be either local or remote.

-policysvr *policy_server_name*
Specifies the name of the policy server.

Comments

After the successful configuration of a Tivoli Access Manager Java application server, SvrSslCfg creates a user account and server entries representing the Java application server in the Tivoli Access Manager user registry. In addition, SvrSslCfg creates a configuration file and a Java key store file, which securely stores a client certificate, locally on the application server. This client certificate permits callers to make authenticated use of Tivoli Access Manager services. Conversely, reconfiguration removes the user and server entries from the user registry and cleans up the local configuration and keystore files.

The contents of an existing configuration file can be modified by using the SvrSslCfg utility. The configuration file and the key store file must already exist when calling SvrSslCfg with all options other than -action config or -action unconfig.

The following options are parsed and processed into the configuration file, but are otherwise ignored in this version of Tivoli Access Manager:

The host name is used to build a unique name (identity) for the application. The pdadmin user list command displays the application identity name in the following format:

server_name/host_name

Note that the pdadmin server list command displays the server name in a slightly different format:

server_name-host_name

```
CLASSPATH=${WAS_HOME}/java/jre/lib/ext/PD.jar:${WAS_CLASSPATH}
java \
-cp ${CLASSPATH} \
-Dpd.cfg.home= ${WAS_HOME}/java/jre \
-Dfile.encoding=ISO8859-1 \
-Dws.output.encoding=CP1047 \
```

```
-Xnoargsconversion \
  com.tivoli.pd.jcfg.SvrSslCfg \
-action config \
-admin_id sec_master \
-admin_pwd $TAM_PASSWORD \
-appsvr_id $APPSVR_ID \
-policysvr ${TAM_HOST}:7135:1 \
-port 7135 \
-authzsvr ${TAM_HOST}:7136:1 \
-mode remote \
-cfg_file ${CFG_FILE} \
-key_file ${KEY_FILE} \
-cfg_action create
```

Creating a trusted user account in Tivoli Access Manager:

Tivoli Access Manager trust association interceptors require the creation of a trusted user account in the shared LDAP user registry.

This account includes the ID and password that WebSEAL uses to identify itself to WebSphere Application Server. To prevent potential vulnerabilities, do not use the sec_master ID as the trusted user account and ensure that the password you use is unique and generated randomly. Use the trusted user account should for the TAI or TAI++ only.

1. Use either the Tivoli Access Manager pdadmin command-line utility or Web Portal Manager to create the trusted user. For example, from the **pdadmin** command line.
2. Reference the code listed below as an example for creating a trusted user account.
3. Reference the following additional resources for more information:
   a. “Configuring WebSEAL for use with WebSphere Application Server”
   b. “Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server” on page 1276

```
pdadmin> user create webseal_userid webseal_userid_DN firstname
         surname password

pdadmin> user modify webseal_userid account-valid yes
```

Configuring WebSEAL for use with WebSphere Application Server:

Use this topic to set the SSO password in WebSEAL for single sign-on to WebSphere Application Server.

A junction must be created between WebSEAL and WebSphere Application Server. This junction carries the iv-credentials (for TAI++) or iv-user (for TAI) and the HTTP basic authentication headers with the request. You can configure WebSEAL to pass the end user identity in other ways, the iv-credentials header is the only one supported by the TAI++ and the iv-user is the only one supported by TAI.

We recommend that communications over the junction use Secure Sockets Layer (SSL) for increased security. Setting up SSL across this junction requires that you configure the HTTP Server used by WebSphere Application Server, and WebSphere Application Server itself, to accept inbound SSL traffic and route it correctly to WebSphere Application Server. This activity requires importing the necessary signing certificates into the WebSEAL certificate keystore, and possibly also the HTTP Server certificate keystore.

Create the junction between WebSEAL and WebSphere Application Server using the **-c iv_creds** option for TAI++ and **-c iv_user** for TAI. Enter either of the following commands as one line using the variables that are appropriate for your environment:

TAI++

```
server task webseald-server create -t ssl -b supply -c iv_creds
 -h host_name -p websphere_app_port_number junction_name
```

TAI

```
server task webseald-server create -t ssl -b supply -c iv_user
 -h host_name -p websphere_app_port_number junction_name
```

Notes:

1. If warning messages are displayed about the incorrect setup of certificates and key databases, delete the junction, correct problems with the key databases, and recreate the junction.
2. The junction can be created as `-t tcp` or `-t ssl`, depending on your requirements.

For single sign-on (SSO) to WebSphere Application Server the SS) password must be set in WebSEAL. To set the password, complete the following steps:

1. Edit the WebSEAL configuration file *webseal_install_directory*/etc/webseald-default.conf Set the following parameter: `basicauth-dummy-passwd=`*webseal_userid_passwd*

   where *webseal_userid_passwd* is the SSO password for the trusted user account set in “Creating a trusted user account in Tivoli Access Manager” on page 1275.
2. Restart WebSEAL.

For more details and options about how to configure junctions between WebSEAL and WebSphere Application Server, including other options for specifying the WebSEAL server identity, refer to the *Tivoli Access Manager WebSEAL Administration Guide* as well as to the documentation for the HTTP Server you are using with your WebSphere Application Server. Tivoli Access Manager documentation is available at http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.

Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server:

Tivoli Access Manager plug-in for Web servers can be used as a security gateway for your protected WebSphere Application Server resources.

With such an arrangement the plug-in authorizes all user requests before passing the credentials of the authorized user to WebSphere Application Server in the form of an iv-creds header. Trust between the plug-in and WebSphere Application Server is established through use of basic authentication headers containing the single sign-on (SSO) user password.

1. The Tivoli Access Manager plug-in for Web servers configuration shows IV headers configured for post-authorization processing, and basic authentication that is configured as the authentication mechanism and for post-authorization processing, as shown in the example below.
2. After a request is authorized, the basic authentication header is removed from the request (`strip-hdr=always`) and a new one is added (`add-hdr=supply`).
3. Included in this new header is the password that is set when the SSO user is created in “Creating a trusted user account in Tivoli Access Manager” on page 1275.
4. Specify this password in the **supply-password** parameter and is passed in the newly created header. This basic authentication header enables trust between WebSphere Application Server and the plug-in.
5. An iv-creds header is also added (`generate=iv-creds`), which contains the credential information of the user passed onto WebSphere Application Server. Session cookies are used to maintain session state.

```
[common-modules]
authentication = BA
session = session-cookie
post-authzn = BA
post-authzn = iv-headers

[iv-headers]
accept = all
```

```
generate = iv-creds

[BA]
strip-hdr = always
add-hdr = supply
supply-password = sso_user_password
```

"Configuring single sign-on using the trust association interceptor" or "Configuring single sign-on using trust association interceptor ++" on page 1278

Configuring single sign-on using the trust association interceptor:

This task is performed to enable single sign-on using the trust association interceptor. These steps involve setting up trust association and creating the interceptor properties.

The following steps are required when setting up security for the first time. Ensure that Lightweight Third Party Authentication (LTPA) is the active authentication mechanism:

1. From the WebSphere Application Server console click **Security > Global security**.
2. Ensure that the Active authentication mechanism field is set to **Lightweight Third Party Authentication (LTPA**). If not, set it and save your changes.

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. You can configure LTPA prior to configuring single sign-on (SSO) by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Although you can use Simple WebSphere Authentication Mechanism (SWAM) by selecting the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel, single sign-on (SSO) requires LTPA as the configured authentication mechanism.

1. From the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.
2. Under Web security, click **Trust association**.
3. Select the **Enable trust association** option.
4. Under Additional properties, click the **Interceptors** link.
5. Click **com.ibm.ws.security.web.WebSealTrustAssociationInterceptor** to use the WebSEAL interceptor. This interceptor is the default.
6. Under Additional properties, click **Custom Properties**.
7. Click **New** to enter the property name and value pairs. Ensure the following parameters are set:

Table 17. Trust association interceptor properties

Option	Description
`com.ibm.websphere.security.trustassociation.types`	Ensure that *webseal* is listed.
`com.ibm.websphere.security.webseal.loginId`	The WebSEAL trusted user as created in "Creating a trusted user account in Tivoli Access Manager" on page 1275 The format of the username is the short name representation. This property is mandatory. If the property is not set in the WebSphere Application Server, TAI initialization fails.
`com.ibm.websphere.security.webseal.id`	The *iv-user* header, which is com.ibm.websphere.security.webseal.id=iv-user

Table 17. Trust association interceptor properties (continued)

Option	Description
com.ibm.websphere.security.webseal.hostnames	Do not set this property if using Tivoli Access Manager plug-in for Web servers. The host names (case sensitive) are trusted and expected in the request header. For example: com.ibm.websphere.security.webseal.hostnames=host1 This includes the proxy host names unless the com.ibm.websphere.security.webseal.ignoreProxy is set to *true*. Obtain a list of servers using the server list **pdadmin** command.
com.ibm.websphere.security.webseal.ports	Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. The corresponding port number of the host names that are expected are in the request header. This includes the proxy ports unless the com.ibm.websphere.security.webseal.ignoreProxy is set to *true*. For example: com.ibm.websphere.security.webseal.ports=80,443
com.ibm.websphere.security.webseal.ignoreProxy	An optional property that if set to *true* or *yes* ignores the proxy host names and ports in the IV header. By default this property is set to *false*.

8. Click **OK**.
9. Save the configuration and log out.
10. Restart WebSphere Application Server.

Configuring single sign-on using trust association interceptor ++:

Perform this task to enable single sign-on using trust association interceptor ++. The steps involve setting up trust association and creating the interceptor properties.

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. However, you may need to configure LTPA prior to configuring the TAMTrustAssociationInterceptorPlus. LTPA is the required authentication mechanism for all trust association interceptors. You can configure LTPA by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.

Note: Enabling Web security single sign-on (SSO) is optional when you configure the TAMTrustAssociationInterceptorPlus. For more information, see “Implementing single sign-on to minimize Web user authentications” on page 1245.

Although you can use Simple WebSphere Authentication Mechanism (SWAM) by selecting the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel, single sign-on (SSO) requires LTPA as the configured authentication mechanism.

1. From the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.
2. Under Web security, click **Trust association**.
3. Click **Enable Trust Association**.
4. Click **Interceptors**.
5. Click **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** to use the WebSEAL interceptor. This interceptor is the default.
6. Click **Custom Properties**.

7. Click **New** to enter the property name and value pairs. Verify that the following parameters are set:

Table 18. Custom properties

Option	Description
com.ibm.websphere.security.webseal.checkViaHeader	You can configure TAI so that the via header can be ignored when validating trust for a request. Set this property to *false* if none of the hosts in the via header need to be trusted. When set to *false* you do not need to set the trusted host names and host ports properties. The only mandatory property to check when via header is *false* is com.ibm.websphere.security.webseal.loginId. The default value of the check via header property is *false*. When using Tivoli Access Manager plug-in for Web servers, set this property to *false*. **Note:** The via header is part of the standard HTTP header that records the server names the request that passed through.
com.ibm.websphere.security.webseal.loginId	The WebSEAL trusted user as created in "Creating a trusted user account in Tivoli Access Manager" on page 1275 The format of the username is the short name representation. This property is mandatory. If it is not set in WebSphere Application Server, the TAI initialization fails.
com.ibm.websphere.security.webseal.id	A comma-separated list of headers that exists in the request. If all of the configured headers do not exist in the request, trust cannot be established. The default value for the ID property is *iv-creds*. Any other values set in WebSphere Application Server are added to the list along with iv-creds, separated by commas.
com.ibm.websphere.security.webseal.hostnames	Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. The property specifies the host names (case sensitive) that are trusted and expected in the request header. Requests arriving from un-listed hosts might not be trusted. If the checkViaHeader property is not set or is set to false then the trusted host names property has no influence. If the checkViaHeader property is set to `true`, and the trusted host names property is not set, TAI initialization fails.
com.ibm.websphere.security.webseal.ports	Do not set this property if using Tivoli Access Manager plug-in for Web servers. This property is a comma-separated list of trusted host ports. Requests that arrive from unlisted ports might not be trusted. If the checkViaHeader property is not set, or is set to `false` this property has no influence. If the checkViaHeader property is set to `true`, and the trusted host ports property is not set in WebSphere Application Server, the TAI initialization fails.
com.ibm.websphere.security.webseal.viaDepth	A positive integer that specifies the number of source hosts in the via header to check for trust. By default, every host in the via header is checked, and if any host is not trusted, trust cannot be established. The via depth property is used when only some of the hosts in the via header have to be trusted. The setting indicates the number of hosts that are required to be trusted. As an example, consider the following header: `Via: HTTP/1.1 webseal1:7002, 1.1 webseal2:7001` If the viaDepth property is not set, is set to 2 or is set to 0, and a request with the previous via header is received then both `webseal1:7002` and `webseal2:7001` need to be trusted. The following configuration applies: `com.ibm.websphere.security.webseal.hostnames = webseal1,webseal2` `com.ibm.websphere.security.webseal.ports = 7002,7001` If the via depth property is set to 1, and the previous request is received, then only the last host in the via header needs to be trusted. The following configuration applies: `com.ibm.websphere.security.webseal.hostnames = webseal2` `com.ibm.websphere.security.webseal.ports = 7001` The viaDepth property is set to `0` by default, which means all of the hosts in the via header are checked for trust.

Table 18. Custom properties (continued)

Option	Description
com.ibm.websphere.security.webseal.ssoPwdExpiry	After trust is established for a request, the single sign-on user password is cached, eliminating the need to have the TAI re-authenticate the single sign-on user with Tivoli Access Manager for every request. You can modify the cache timeout period by setting the single sign-on password expiry property to the required time in seconds. If the password expiry property is set to 0, the cached password never expires. The default value for the password expiry property is 600.
com.ibm.websphere.security.webseal.ignoreProxy	This property can be used to tell the TAI to ignore proxies as trusted hosts. If set to true the comments field of the hosts entry in the via header is checked to determine if a host is a proxy. Remember that not all proxies insert comments in the via header indicating that they are proxies. The default value of the ignoreProxy property is false. If the checkViaHeader property is set to false then the ignoreProxy property has no influence in establishing trust.
com.ibm.websphere.security.webseal.configURL	For the TAI to establish trust for a request, it requires that the SvrSslCfg run for the Java Virtual Machine on the Application Server and result in the creation of a properties file. If this properties file is not at the default URL, which is file://java.home/PdPerm.properties, the correct URL of the properties file must be set in the configuration URL property. If this property is not set, and the SvrSslCfg-generated properties file is not in the default location, the TAI initialization fails. The default value for the config URL property is file://${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties.

8. Click **OK**.
9. Save the configuration and log out.
10. Restart WebSphere Application Server.

Configuring global sign-on principal mapping:

You can create a new application login that uses the Tivoli Access Manager GSO database to store the login credentials.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > Application logins**.
3. Click **New** to create a new Java Authentication and Authorization Service (JAAS) login configuration.
4. Enter the alias name of the new application login. Click **Apply**.
5. Under Additional properties, click **JAAS login modules** to define the JAAS Login Modules.
6. Click **New** and enter the following information:

   Module class name: com.tivoli.pdwas.gso.AMPrincipalMapper
   Use Login Module Proxy: enable
   Authentication strategy: REQUIRED
7. Click **Apply**
8. Under Additional Properties section, click **Custom Properties** to define login module-specific values that are passed directly to the underlying login modules.
9. Click **New**.

   The Tivoli Access Manager principal mapping module uses the authDataAlias configuration string to retrieve the correct user name and password from the security configuration.

   The authDataAlias attribute that is passed to the module is configured for the J2C connection factory. Because the authDataAlias attribute is an arbitrary string that is entered at configuration time, the following scenarios are possible:

- The authDataAlias attribute contains both the global sign-on (GSO) resource name and the user name. The format of this string is `"Resource/User"`.
- The authDataAlias attribute contains the GSO Resource name only. The user name is determined by using the Subject of the current session.

The scenario to use is determined by a JAAS configuration option, as shown here:

Name: com.tivoli.pd.as.gso.AliasContainsUserName

Value: True, if the alias contains the user name; false, if the user name must be retrieved from the security context

When entering `authDataAlias` attributes through the WebSphere Application Server administrative console, the node name is automatically pre-pended to the alias. The JAAS configuration entry determines whether this node name is removed or included as part of the resource name, as shown here:

Name: com.tivoli.pd.as.gso.AliasContainsNodeName

Value: True, if the alias contains the node name

Note: If the `PdPerm.properties` configuration file is not located in the `JAVA_HOME/PdPerm.properties` default location, then you also need to add the following property:

Name: com.tivoli.pd.as.gso.AMCfgURL

Value: file:///*path to PdPerm.properties*

Enter each new parameter using the following scenario information as a guide, then click **Apply**.

Scenario 1
Auth Data Alias - BackendEIS/eisUser
Resource - BackEndEIS
User - eisUser
Principal Mapping Parameters

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	true
com.tivoli.pd.as.gso.AliasContainsNodeName	false
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 2
Auth Data Alias - BackendEIS
Resource - BackEndEIS
User - Currently authenticated WebSphere Application Server user
Principal Mapping Parameters

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	false
com.tivoli.pd.as.gso.AliasContainsNodeName	false
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 3
Auth Data Alias - nodename/BackendEIS/eisUser

Resource - BackEndEIS
User - eisUser
Principal Mapping Parameters

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	true
com.tivoli.pd.as.gso.AliasContainsNodeName	true
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 4
Auth Data Alias - nodename/BackendEIS/eisUser
Resource - nodename/BackEndEIS (notice that node name is not removed)
User - eisUser
Principal Mapping Parameters

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	true
com.tivoli.pd.as.gso.AliasContainsNodeName	false
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 5
Auth Data Alias - BackendEIS/eisUser
Resource - BackEndEIS
User - eisUser
Principal Mapping Parameters

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	false
com.tivoli.pd.as.gso.AliasContainsNodeName	true
com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

Scenario 6
Auth Data Alias - nodename/BackendEIS/eisUser
Resource - nodename/BackendEIS/eisUser
(notice that the resource is the same as Auth Data Alias).
User - Currently authenticated WebSphere Application Server user
Principal Mapping Parameters

Name	Value
delegate	com.tivoli.pdwas.gso.AMPrincipalMapper
com.tivoli.pd.as.gso.AliasContainsUserName	false
com.tivoli.pd.as.gso.AliasContainsNodeName	false

com.tivoli.pd.as.gso.AMLoggingURL	file:///*jlog_props_path*
debug	false

10. Create the Java 2 Connector (J2C) authentication aliases. The user name and password that are assigned to these alias entries are irrelevant because Tivoli Access Manager is responsible for providing user names and passwords. However, the user name and password that are assigned to the J2C authentication aliases need to exist so that they can be selected for the J2C connection factory in the administrative console.

    To create the J2C authentication aliases, from the WebSphere Application Server administrative console, click **Security** >**Secure administration, applications, and infrastructure**. Under Authentication, click **Java Authentication and Authorization Service** > **J2C authentication data**, and then click **New** for each new entry. Refer to the previous table for scenario inputs.

    The connection factories for each resource adapter that need to use the GSO database must be configured to use the Tivoli Access Manager Principal mapping module:

    a. From the WebSphere Application Server administrative console, click **Applications > Enterprise Applications** > *application_name* **> Resourcer references**. Note that J2C connection factories must be already configured for the selected application. To configure a new J2C connection factory, see tdat_confconfac.dita.

    b. Under Additional properties, click **Resource Adapter**.

       The resource adapter can be standalone and does not need to be packaged with the application. The resource adapter is configured from **Resources** > **Resource Adapters** for standalone scenarios.

    c. Under Additional properties, click **J2C Connection Factories**.

    d. Click **New** and enter the connection factory properties.

    e. When finished, click **Apply > Save**.

    Attention:
    Custom mapping configuration for the connection factory is deprecated in WebSphere Application Server Version 6. To configure the GSO credential mapping, use the Map Resource References to Resources panel on the administrative console. For more information, see “J2EE connector security” on page 682.

Propagating security attributes among application servers

Use the security attribute propagation feature of WebSphere Application Server to send security attribute information regarding the original login to other servers using a token. This topic will help to configure WebSphere Application Server to propagate security attributes to other servers.

To fully enable security attribute propagation, you must configure the single sign-on (SSO), Common Secure Interoperability Version 2 (CSIv2) inbound, and CSIv2 outbound panels in the WebSphere Application Server administrative console. You can enable just the portions of security attribute propagation relevant to your configuration. For example, you can enable Web propagation, which is propagation amongst front-end application servers, using either the push technique (DynaCache) or the pull technique (remote method to originating server).

You also can choose whether to enable Remote Method Invocation (RMI) outbound and inbound propagation, which is commonly called downstream propagation. Typically both types of propagation are enabled for any given cell. In some cases, you might want to choose a different option for a specific application server using the server security panel within the specific application server settings.

To access the server security panel in the administrative console, click **Servers > Application Servers > *server_name***. Under Security, click **Server security**.

Complete the following steps to configure WebSphere Application Server for security attribute propagation:

1. Access the WebSphere Application Server administrative console by typing `http://server_name:port_number/ibm/console`. The administrative console address might differ if you have previously changed the port number.
2. Click **Security > Secure administration, applications, and infrastructure**.
3. Under Web security, click **Single sign-on (SSO)**.
4. **Optional:** Select the **Interoperability Mode** option if you need to interoperate with servers that do not support security attribute propagation. Servers that do not support security attribute propagation receive the Lightweight Third Party Authentication (LTPA) token and the Propagation token, but ignore the security attribute information that they do not understand.
5. Select the **Web inbound security attribute propagation** option. The Web inbound security attribute propagation option enables horizontal propagation, which allows the receiving SSO token to retrieve the login information from the original login server. If you do not enable this option, downstream propagation can occur if you enable the Security Attribute Propagation option on both the CSIv2 Inbound authentication and CSIv2 outbound authentication panels.

   Typically, you enable the Web inbound security attribute propagation option if you need to gather dynamic security attributes set at the original login server that cannot be regenerated at the new front-end server. These attributes include any custom attributes that might be set in the PropagationToken token using the com.ibm.websphere.security.WSSecurityHelper application programming interfaces (APIs). You must determine whether enabling this option improves or degrades the performance of your system. While the option prevents some remote user registry calls, the deserialization and decryption of some tokens might impact performance. In some cases propagation is faster, especially if your user registry is the bottleneck of your topology. It is recommended that you measure the performance of your environment both using and not using this option. When you test the performance, it is recommended that you test in the operating environment of the typical production environment with the typical number of unique users accessing the system simultaneously.
6. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 inbound authentication**. The Login configuration field specifies `RMI_INBOUND` as the system login configuration that is used for inbound requests. To add custom Java Authentication and Authorization Service (JAAS) login modules, complete the following steps:
   a. Click **Security > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **System logins**. A list of the system login configurations is displayed. WebSphere Application Server provides the following pre-configured system login configurations: DEFAULT, LTPA, LTPA_WEB, RMI_INBOUND, RMI_OUTBOUND, SWAM, WEB_INBOUND, wssecurity.IDAssertion, and wssecurity.Signature. Do not delete these predefined configurations.

      Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.
   b. Click the name of the login configuration that you want to modify.
   c. Under Additional Properties, click **JAAS Login Modules**. The JAAS Login Modules panel is displayed, which lists all of the login modules that are processed in the login configuration. Do not delete the required JAAS login modules. Instead, you can add custom login modules before or after the required login modules. If you add custom login modules, do not begin their names with com.ibm.ws.security.server.

      You can specify the order in which the login modules are processed by clicking **Set Order**.
7. Select the **Security attribute propagation** option on the CSIv2 inbound authentication panel. When you select **Security Attribute Propagation**, the server advertises to other application servers that it can receive propagated security attributes from another server in the same realm over the Common Secure Interoperability version 2 (CSIv2) protocol.
8. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 Outbound authentication**. The CSIv2 outbound authentication panel is displayed. The **Login configuration** field specifies `RMI_OUTBOUND` as the JAAS login configuration that is used for

outbound configuration. You cannot change this login configuration. Instead, you can customize this login configuration by completing the substeps that are listed previously for CSIv2 Inbound authentication.

9. **Optional:** Verify that the **Security Attribute Propagation** option is selected if you want to enable outbound Subject and security context token propagation for the Remote Method Invocation (RMI) protocol. When you select this option, WebSphere Application Server serializes the Subject contents and the PropagationToken contents. After the contents are serialized, the server uses the CSIv2 protocol to send the Subject and PropagationToken token to the target servers that support security attribute propagation. If the receiving server does not support security attribute tokens, WebSphere Application Server sends the Lightweight Third Party Authentication (LTPA) token only.

   Important: WebSphere Application Server propagates only the objects within the Subject that it can serialize. The server propagates custom objects on a best-effort basis.

   When **Security Attribute Propagation** is enabled, WebSphere Application Server adds marker tokens to the Subject to enable the target server to add additional attributes during the inbound login. During the commit phase of the login, the marker tokens and the Subject are marked as read-only and cannot be modified thereafter.
10. **Optional:** Select the **Custom Outbound Mapping** option if you clear the **Security Attribute Propagation** option and you want to use the RMI_OUTBOUND login configuration. If neither the **Custom Outbound Mapping** option nor the **Security Attribute Propagation** option is selected, WebSphere Application Server does not call the RMI_OUTBOUND login configuration. If you need to plug in a credential mapping login module, you must select the **Custom Outbound Mapping** option.
11. **Optional:** Specify trusted target realm names in the **Trusted Target Realms** field. By specifying these realm names, information can be sent to servers that reside outside the realm of the sending server to support inbound mapping that is at these downstream servers. To perform outbound mapping to a realm different from the current realm, you must specify the realm in this field so that you can get to this point without having the request rejected because of a realm mismatch. If you need WebSphere Application Server to propagate security attributes to another realm when a request is sent, you must specify the realm name in the **Trusted Target Realms** field. Otherwise, the security attributes are not propagated to the unspecified realm. You can add multiple target realms by adding a pipe (|) delimiter between each entry.
12. **Optional:** Enable propagation for a pure client. For a pure client to propagate attributes added to the invocation Subject, you must add the following property to the `sas.client.props` file:

    ```
    com.ibm.CSI.rmiOutboundPropagationEnabled=true
    ```

    Note: The `sas.client.props` file is located at `<WAS-HOME>/profiles/<ProfileName>/properties>`.

After completing these steps, you have configured WebSphere Application Server to propagate security attributes to other servers.

If you need to disable security attribute propagation, determine whether you need to disable it for either the server level or the cell level.

Attention: Changes to the server-level settings override the cell settings.

To disable security attribute propagation on the server level, complete the following steps:

1. Click **Server > Application Servers > *server_name***.
2. Under Security, click **Server security**.
3. Select the **RMI/IIOP security for this server overrides cell settings** option.
4. Disable security attribute propagation for inbound requests by clicking **CSI inbound authentication** under Additional Properties and clearing the **Security attribute propagation** option.
5. Disable security attribute propagation for outbound requests by clicking **CSI outbound authentication** under Additional Properties and clearing the **Security attribute propagation** option.

To disable security attribute propagation on the cell level, undo each of the steps that you completed to enable security attribute propagation in this task.

Configuring the authentication cache

The security authentication cache affects the frequency of rehashing and the distribution of the hash algorithms.

To configure the authentication cache properties, complete the following steps:

1. Click **Servers > Application Servers >** ***server_name*** .
2. Under Server infrastructure, click **Java and Process Management > Process definition**.
3. Under Additional properties, click **Java Virtual Machine > Custom Properties**.
4. Click **New** to specify a new custom property.

For information on the supported authentication cache properties, see “Security cache properties.”

Security cache properties

The following Java virtual machine (JVM) security cache custom properties determine whether the authentication cache is enabled or disabled. If the authentication cache is enabled, as recommended, these custom properties specify the initial size of the primary and secondary hash table caches, which affect the frequency of rehashing and the distribution of the hash algorithms.

Important: The com.ibm.websphere.security.util.tokenCacheSize and com.ibm.websphere.security.util.LTPAValidationCacheSize properties were replaced with the com.ibm.websphere.security.util.authCacheSize property.

You can specify these system properties by completing the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Click **Java and Process Management > Process Definition**.
3. Under Additional properties, click **Java Virtual Machine**.
4. Specify the property name and its value in the Generic JVM arguments field. You can specify multiple property name and value pairs delimited by a space.

WebSphere Application Server includes the following security cache custom properties:

com.ibm.websphere.security.util.authCacheEnabled

Specifies whether to disable the authentication cache. It is recommended that you leave the authentication cache enabled for performance reasons. However, you can disable the authentication cache for debug or measurement purposes.

Default:	True

com.ibm.websphere.security.util.authCacheSize

Specifies the initial size of the primary and secondary hash table caches. A higher number of available hash values might decrease the occurrence of hash collisions. A hash collision results in a linear search for the hash bucket, which might decrease the retrieval time. If several entries compose a hash table cache, you create a table with a larger capacity that supports more efficient hash entries instead of allowing automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

Default:	200
Type:	Integer

Configuring IIOP authentication

WebSphere Application Server enables you to specify Internet Inter-ORB Protocol (IIOP) authentication for both inbound and outbound authentication requests. For inbound requests, you can specify the type of accepted authentication, such as basic authentication. For outbound requests, you can specify properties such as type of authentication, identity assertion or login configurations that are used for requests to downstream servers.

The following topics are covered in this section:

- Configuring Common Secure Interoperability Version 2 inbound authentication
- Configuring Common Secure Interoperability Version 2 outbound authentication

Configuring Common Secure Interoperability Version 2 inbound authentication

Inbound authentication refers to the configuration that determines the type of accepted authentication for inbound requests. This authentication is advertised in the interoperable object reference (IOR) that the client retrieves from the name server.

1. Start the administrative console.
2. Click **Security > Secure administration, applications, and infrastructure**.
3. Under RMI/IIOP security, click **CSIv2 inbound authentication**.
4. Consider the following layers of security:
   - Identity assertion (attribute layer).

     When selected, this server accepts identity tokens from upstream servers. If the server receives an identity token, the identity is taken from an originating client. For example, the identity is in the same form that the originating client presented to the first server. An upstream server sends the identity of the originating client. The format of the identity can be either a principal name, a distinguished name, or a certificate chain. In some cases, the identity is anonymous. It is important to trust the upstream server that sends the identity token because the identity authenticates on this server. Trust of the upstream server is established either using Secure Sockets Layer (SSL) client certificate authentication or basic authentication. You must select one of the two layers of authentication in both inbound and outbound authentication when you choose identity assertion.

     The server ID is sent in the client authentication token with the identity token. The server ID is checked against the trusted server ID list. If the server ID is on the trusted server list, the server ID is authenticated. If the server ID is valid, the identity token is put into a credential and used for authorization of the request.

     For more information, refer to Identity assertion.
   - User ID and password (message layer).

     This type of authentication is the most typical. The user ID and password or authenticated token is sent from a pure client or from an upstream server. However, the upstream server cannot be a z/OS server because z/OS does not support a user ID or password from a server acting as a client. When a user ID and password are received at the server, they are authenticated with the user registry.

     Usually, a token is sent from an upstream server and a user ID and password are sent from a client, including a servlet. When a token is received at the server level, the token is validated to determine whether tampering has occurred or whether it is expired.

     For more information, refer to User ID and password.
   - Secure Sockets Layer client certificate authentication (transport layer).

     The SSL client certificate is used to authenticate instead of using user ID and Password. If a server delegates an identity to a downstream server, the identity comes from either the message layer (a client authentication token) or the attribute layer (an identity token), and not from the transport layer through the client certificate authentication.

     A client has an SSL client certificate that is stored in the keystore file of the client configuration. When SSL client authentication is enabled on this server, the server requests that the client send the SSL client certificate when the connection is established. The certificate chain is available on the socket whenever a request is sent to the server. The server request interceptor gets the certificate

chain from the socket and maps this certificate chain to a user in the user registry. This type of authentication is optimal for communicating directly from a client to a server. However, when you have to go downstream, the identity typically flows over the message layer or through identity assertion.

5. Consider the following points when deciding what type of authentication to accept:
   - A server can receive multiple layers simultaneously, so an order of precedence rule decides which identity to use. The identity assertion layer has the highest priority, the message layer follows, and the transport layer has the lowest priority. The SSL client certificate authentication is used when it is the only layer provided. If the message layer and the transport layer are provided, the message layer is used to establish the identity for authorization. The identity assertion layer is used to establish precedence when provided.
   - Does this server usually receive requests from a client, from a server, or both? If the server always receives requests from a client, identity assertion is not needed. You can choose either the message layer, the transport layer, or both. You also can decide when authentication is required or just supported. To select a layer as required, the sending client must supply this layer, or the request is rejected. However, if the layer is only supported, the layer might not be supplied.
   - What kind of client identity is supplied? If the client identity is client certificates authentication and you want the certificate chain to flow downstream so that it maps to the downstream server user registries, identity assertion is the appropriate choice. Identity assertion preserves the format of the originating client. If the originating client authenticated with a user ID and password, a principal identity is sent. If authentication is done with a certificate, the certificate chain is sent.

     In some cases, if the client authenticated with a token and a Lightweight Directory Access Protocol (LDAP) server is the user registry, then a distinguished name (DN) is sent.
6. Configure a trusted server list. When identity assertion is selected for inbound requests, insert a pipe-separated (I) list of server administrator IDs to which this server can support identity token submission. For backwards compatibility, you can still use a comma-delimited list. However, if the server ID is a distinguished name (DN), then you must use a pipe-delimited (|) list because a comma delimiter does not work. If you choose to support any server sending an identity token, you can enter an asterisk (*) in this field. This action is called *presumed trust*. In this case, use SSL client certificate authentication between servers to establish the trust.
7. Configure session management. You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between a client and server is authenticated. All subsequent requests (or until the credential token expires) reuse the session information, including the credential. A client sends a context ID for subsequent requests. The context ID is scoped to the connection for uniqueness.

When you finish configuring this panel, you have configured most of the information that a client gathers when determining what to send to this server. A client or server outbound configuration with this server inbound configuration, determines the security that is applied. When you know what clients send, the configuration is simple. However, if you have a diverse set of clients with differing security requirements, your server considers various layers of authentication.

For a J2EE application server, the authentication choice is usually either identity assertion or message layer because you want the identity of the originating client delegated downstream. You cannot easily delegate a client certificate using an SSL connection. It is acceptable to enable the transport layer because additional server security, as the additional client certificate portion of the SSL handshake, adds some overhead to the overall SSL connection establishment.

After you determine which type of authentication data this server might receive, you can determine what to select for outbound security. For more information, see Configuring Common Secure Interoperability Version 2 outbound authentication.

Common Secure Interoperability inbound authentication settings:

Use this page to specify the features that a server supports for a client accessing its resources.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RMI/IIOP security > CSIv2 inbound authentication**.

You can also view this administrative console page on the server page by completing the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Server security**.
3. Under Additional properties, click **CSIv2 inbound authentication**.

Use common secure interoperability (CSI) inbound authentication settings for configuring the type of authentication information that is contained in an incoming request or transport.

Authentication features include three layers of authentication that you can use simultaneously:

- **Transport layer**. The transport layer, which is the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.
- **Message layer**. The message layer might contain a user ID and password or an authenticated token with an expiration.
- **Attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The identity layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information that is presented during the request. The client picks up the interoperable object reference (IOR) from the namespace and reads the values from the tagged component to determine what the server needs for security.

Basic authentication:

Specifies that basic authentication occurs over the message layer.

Basic authentication occurs in the message layer. This type of authentication typically involves sending a user ID and a password from the client to the server for authentication.

This authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable, for example, Lightweight Third Party Authentication (LTPA).

If you click **Basic Authentication** and LTPA is the configured authentication protocol, user name, password, and LTPA tokens are accepted.

The following options are available for Basic Authentication:

Never This option indicates that this server cannot accept user ID and password authentication.

Supported
This option indicates that a client communicating with this server can specify a user ID and password. However, a method might be invoked without this type of authentication. For example, an anonymous or client certificate might be used instead.

Required
This option indicates that clients communicating with this server must specify a user ID and password for any method request.

Basic authentication takes precedence over client certificate authentication, if both are performed.

Client certificate authentication:

Specifies that authentication occurs when the initial connection is made between the client and the server during a method request.

In the transport layer, Secure Sockets Layer (SSL) client certificate authentication occurs. In the message layer, basic authentication (user ID and password) is performed. Client certificate authentication typically performs better than message layer authentication, but requires some additional setup. These additional steps involve verifying that the server trusts the signer certificate of each client to which it is connected. If the client uses a certificate authority (CA) to create its personal certificate, you only need the CA root certificate in the server signer section of the SSL trust file.

When the certificate is authenticated to a Lightweight Directory Access Protocol (LDAP) user registry, the distinguished name (DN) is mapped based on the filter that is specified when configuring LDAP. When the certificate is authenticated to a local OS user registry, the first attribute of the distinguished name (DN) in the certificate, which is typically the common name, is mapped to the user ID in the registry.

The identity from client certificates is used only if no other layer of authentication is presented to the server.

Never This option indicates that clients cannot attempt Secure Sockets Layer (SSL) client certificate authentication with this server.

Supported
This option indicates that clients connecting to this server can authenticate using SSL client certificates. However, the server can invoke a method without this type of authentication. For example, anonymous or basic authentication can be used instead.

Required
This option indicates that clients connecting to this server must authenticate using SSL client certificates before invoking the method.

Trusted identities:

Specifies a pipe-separated (|) list of trusted server administrator user IDs, which are trusted to perform identity assertion to this server. For example, `serverid1|serverid2|serverid3`. The application server supports the comma (,) character as the list delimiter for backwards compatibility. The application server checks the comma character when the pipe character (|) fails to find a valid trusted server ID.

Use this list to decide whether a server is trusted. Even if the server is on the list, the sending server must still authenticate with the receiving server to accept the identity token of the sending server.

Data type String

Stateful sessions:

Select this option to enable stateful sessions, which are used mostly for performance improvements.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and the ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. Whenever the security session is not valid and the authentication retry is enabled, which is the default, the client-side security interceptor invalidates the client-side session and submits the request again without user awareness. This situation might occur if the session does not exist on the server (the server failed and resumed operation). When this value is disabled, every method invocation must authenticate again.

Data type String

Login configuration:

Specifies the type of system login configuration to use for inbound authentication.

You can add custom login modules by clicking **Security > Secure administration, applications, and infrastructure**. Under Authentication, click **Java Authentication and Authorization Service > System logins**.

Security attribute propagation:

Select this option to support security attribute propagation during login requests. When you select this option, the application server retains additional information about the login request, such as the authentication strength used, and retains the identity and location of the request originator.

Verify that you are using Lightweight Third Party Authentication (LTPA) as your authentication mechanism. LTPA is the only authentication mechanism supported when you enable the security attribute propagation feature.

To configure LTPA, click **Security > Secure administration, applications, and infrastructure**. Under Authentication, click **Authentication mechanisms and expiration**.

If you do not select this option, the application server does not accept any additional login information to propagate to downstream servers.

Configuring Common Secure Interoperability Version 2 outbound authentication

The following choices are available when configuring the Common Secure Interoperability Version 2 (CSIv2) Outbound Authentication panel.

Outbound authentication refers to the configuration that determines the type of authentication that is performed for outbound requests to downstream servers. Several *layers* or *methods* of authentication can occur. The downstream server inbound authentication configuration must support at least one choice made in this server outbound authentication configuration. If nothing is supported, the request might go outbound as unauthenticated. This situation does not create a security problem because the authorization runtime is responsible for preventing access to protected resources. However, if you choose to prevent an unauthenticated credential from going outbound, you might want to designate one of the authentication layers as required, rather than supported. If a downstream server does not support authentication, then when authentication is required, the method request fails to go outbound.

The following choices are available in the Common Secure Interoperability Version 2 (CSIv2) Outbound Authentication panel. Remember that you are not required to complete these steps in the displayed order. Rather, these steps are provided to help you understand your choices for configuring outbound authentication.

- Select **Identity Assertion** (attribute layer). When selected, this server sends an identity token to a downstream server if the downstream server supports identity assertion. When an originating client authenticates to this server, the authentication information supplied is preserved in the outbound identity token. If the client authenticating to this server uses client certificate authentication, then the identity token format is a certificate chain, containing the exact client certificate chain from the inbound socket. The same scenario is true for other mechanisms of authentication. Read theIdentity Assertion topic for more information.
- Select **User ID** and **Password** (message layer). This type of authentication is the most typical. The user ID and password (if BasicAuth credential) or authenticated token (if authenticated credential) are sent outbound to the downstream server if the downstream server supports message layer authentication in the inbound authentication panel. Refer to the Message Layer Authentication article for more information.
- Select **SSL Client certificate authentication** (transport layer). The main reason to enable outbound Secure Sockets Layer (SSL) client authentication from one server to a downstream server is to create a trusted environment between those servers. For delegating client credentials, use one of the two layers

mentioned previously. However, you might want to create SSL personal certificates for all the servers in your domain, and only trust those servers in your SSL truststore file. No other servers or clients can connect to the servers in your domain, except at the tiers where you want them. This process can protect your enterprise bean servers from access by anything other than your servlet servers.

Configuring session management:
You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between this server and the downstream server is authenticated. All subsequent requests reuse the session information, including the credential. A *unique session entry* is defined as the combination of a unique client authentication token and an identity token, scoped to the connection.

When you finish configuring this panel, you configured the information that this server uses to make decisions about the type of authentication to perform with downstream servers. If the downstream server is configured not to support the outbound configuration of the server, the following exception likely occurs:

```
Exception received: org.omg.CORBA.INITIALIZE:
CWWSA1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:  The client security
configuration (sas.client.props or outbound settings in GUI) does not
support the server security configuration for the following reasons:
ERROR 1: CWWSA0607E: The client requires SSL Confidentiality but the server
does not support it.
ERROR 2: CWWSA0610E: The server requires SSL Integrity but the client does
not support it.
ERROR 3: CWWSA0612E: The client requires client (e.g., userid/password or token),
but the server does not support it.
minor code: 0  completed: No
        at com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.
getConnectionKey(SecurityConnectionInterceptor.java:1770)
        at com.ibm.ws.orbimpl.transport.WSTransport.getConnection(Unknown Source)
        at com.ibm.rmi.iiop.TransportManager.get(TransportManager.java:79)
        at com.ibm.rmi.iiop.GIOPImpl.locate(GIOPImpl.java:167)
        at com.ibm.CORBA.iiop.ClientDelegate._createRequest(ClientDelegate.java:2088)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1264)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1177)
        at com.ibm.CORBA.iiop.ClientDelegate.request(ClientDelegate.java:1726)
        at org.omg.CORBA.portable.ObjectImpl._request(ObjectImpl.java:245)
        at com.ibm.WsnOptimizedNaming._NamingContextStub.get_compatibility_level
(Unknown Source)
        at com.ibm.websphere.naming.DumpNameSpace.getIdlLevel(DumpNameSpace.java:300)
        at com.ibm.websphere.naming.DumpNameSpace.getStartingContext
(DumpNameSpace.java:329)
        at com.ibm.websphere.naming.DumpNameSpace.main(DumpNameSpace.java:268)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:163)
```

The reasons for the mismatch are explained in the exception. You can make the corrections when you configure the outbound configuration for this server, or when you configure the inbound configuration of the downstream server. If multiple reasons exist for a failure, the reasons are explained as message text in the exception.

Typically, the outbound authentication configuration is for an upstream server to communicate with a downstream server. Most likely, the upstream server is a servlet server and the downstream server is an Enterprise JavaBeans (EJB) server. On a servlet server, the client authentication that is performed to access the servlet can be one of many different types of authentication, including client certificate and basic authentication. When receiving basic authentication data, whether through a prompt login or a form-based login, the basic authentication information is typically authenticated to from a credential of the mechanism type that is supported by the server, such as the Lightweight Third Party Authentication (LTPA). When LTPA is the mechanism, a forwardable token exists in the credential. Choose the message

layer (BasicAuth) authentication to propagate the client credentials. If the credential is created using a certificate login and you want to preserve sending the certificate downstream, you might decide to go outbound with identity assertion.

Save the configuration and restart the server for the changes to take effect.

Common Secure Interoperability Version 2 outbound authentication settings:

Use this page to specify the features that a server supports when acting as a client to another downstream server.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RMI/IIOP security > CSIv2 outbound authentication**.

You also can view this administrative console page by completing the following steps:

1. Click **Servers > Application Servers > *server_name***.
2. Under Security, click **Server security**.
3. Click **CSIv2 outbound authentication**.

Authentication features include the following layers of authentication that you can use simultaneously:

Transport layer
The transport layer, the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.

Message layer
The message layer might contain a user ID and password or authenticated token.

Attribute layer
The attribute layer might contain an identity token, which is an identity from an upstream server that is already authenticated. The attribute layer has the highest priority, followed by the message layer and then the transport layer. If this server sends all three - the attribute layer, the message layer, and the transport layer - only the attribute layer is used by the downstream server. The only way to use the SSL client certificate as the identity is if it is the only information presented during the outbound request.

Basic authentication:

Specifies whether to send a user ID and a password from the client to the server for authentication.

This type of authentication occurs over the message layer. Basic authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)). Basic authentication refers to any authentication over the message layer and indicates user ID and password as well as token-based authentication.

The following options are available:

Never This option indicates that this server does not send user ID and password authentication information to downstream servers. By selecting never, requests to downstream servers that require basic authentication fail.

Supported
This option indicates that this server can specify a user ID and password to authenticate with downstream servers. However, a method might be invoked without this type of authentication. For example, the server can use anonymous or client certificate instead.

Required
This option indicates that this server must specify a user ID and password to authenticate with

downstream servers for any method request. This server cannot initiate requests with servers that do not support or require basic authentication for inbound requests.

Client certificate authentication:

Specifies whether a client certificate from the configured keystore is used to authenticate to the server when the SSL connection is made between this server and a downstream server, provided that the downstream server supports client certificate authentication.

Typically, client certificate authentication has a higher performance than message layer authentication, but requires some additional setup. These additional steps include verifying that this server has a personal certificate and that the downstream server has the signer certificate of this server.

If you select client certificate authentication, the following options are available:

Never This option indicates that this server does not attempt Secure Sockets Layer (SSL) client certificate authentication with downstream servers.

Supported
This option indicates that this server can use SSL client certificates to authenticate to downstream servers. However, a method can be invoked without this type of authentication. For example, the server can use anonymous or basic authentication instead.

Required
This option indicates that this server must use SSL client certificates to authenticate to downstream servers.

Identity assertion:

Specifies whether to assert identities from one server to another during a downstream enterprise bean invocation.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the identity specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the identity of the sending server to ensure a trusted identity.

When specifying identity assertion on the CSIv2 authentication outbound panel, you must also select basic authentication as supported or required on the CSIv2 authentication inbound panel. The server identity can then be submitted with the identity token, so that the receiving server can *trust* the sending server. Without specifying basic authentication as supported or required, trust is not established and the identity assertion fails.

Use server trusted identity
Specifies the server identity that the application server uses to establish trust with the target server. The server identity can be sent using one of the following methods:

- A server ID and password when the server password is specified in the registry configuration.
- A server ID in a Lightweight Third Party Authentication (LTPA) token when the internal server ID is used.

For interoperability with application servers other than WebSphere Application Server, use of the following methods:

- Configure the server ID and password in the registry.
- Select the **Specify an alternative trusted identity** option and specify the trusted identity and password so that an interoperable Generic Security Services Username Password (GSSUP) token is sent instead of an LTPA token.

Specify an alternative trusted identity

Specifies an alternative user as the trusted identity that is sent to the target servers instead of sending the server identity. This option is recommended for identity assertion. The identity is automatically trusted when it is sent within the same cell and does not need to be in the trusted identities list within the same cell. However, this identity must be in the registry of the target servers in an external cell and the user ID must be on the trusted identities list or the identity is rejected during trust evaluation.

Trusted identity
Specifies the trusted identity that is sent from the sending server to the receiving server.

If you specify an identity in this field, it can be selected on the panel for your configured user account repository. If you do not specify an identity, a Lightweight Third Party Authentication (LTPA) token is sent between the servers.

Password
Specifies the password that is associated with the trusted identity.

Confirm password
Confirms the password that is associated with the trusted identity.

Stateful sessions:

Specifies whether to reuse security information during authentication. This option is usually used to increase performance.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and that ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. When the security session is not valid and if authentication retry is enabled, which is the default, the client-side security interceptor invalidates the client-side session and resubmits the request transparently. For example, if the session does not exist on the server; the server fails and resumes operation.

When this value is disabled, every method invocation must authenticate again.

Login configuration:

Specifies the type of system login configuration that is used for outbound authentication.

You can add custom login modules before or after this login module by completing the following steps:
1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > System logins > New**.

Custom outbound mapping:

Enables the use of custom Remote Method Invocation (RMI) outbound login modules.

The custom login module maps or performs other functions before the predefined RMI outbound call.

To declare a custom outbound mapping, complete the following steps:
1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > System logins > New**.

Security attribute propagation:

Enables the application server to propagate the Subject and the security content token to other application servers using the Remote Method Invocation (RMI) protocol.

Verify that you are using Lightweight Third Party Authentication (LTPA) as your authentication mechanism. LTPA is the only authentication mechanism that is supported when you enable the security attribute propagation feature. To configure LTPA, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Authentication mechanisms and expiration**.

By default, the Security attribute propagation option is enabled and outbound login configuration is invoked. If you clear this option, the application server does not propagate any additional login information to downstream servers. If you select the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel, the security attribute propagation feature is not supported.

Note: SWAM is deprecated in the application server Version 6.1 and will be removed in a future release.

Trusted target realms:

Specifies a list of trusted target realms, separated by a pipe character (|), that differ from the current realm.

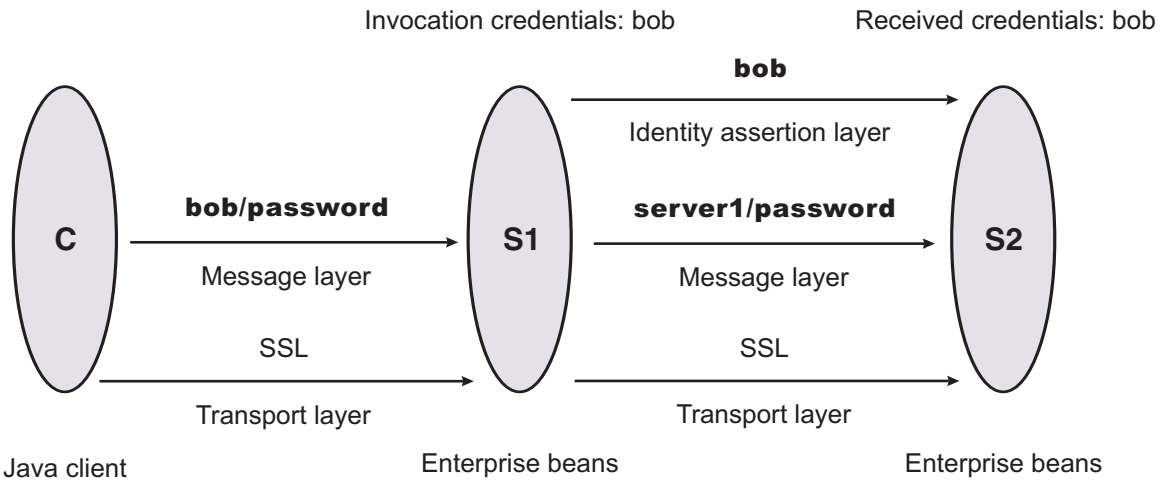
Prior to WebSphere Application Server, Version 5.1.1, if the current realm does not match the target realm, the authentication request is not sent outbound to other application servers.

Example: Common Secure Interoperability Version 2 scenarios

The articles included in this section provide specific scenarios demonstrating how to configure Common Secure Interoperability Version 2 (CSIv2).

Scenario 1: Basic authentication and identity assertion:

This example presents a pure Java client, C, that accesses a secure enterprise bean on server, S1, through user bob. The following steps take you through the configuration of C, S1, and S2.



The enterprise bean code on S1 accesses another enterprise bean on server, S2. This configuration uses identity assertion to propagate the identity of bob to the downstream server, S2. S2 trusts that bob already is authenticated by S1 because it trusts S1. To gain this trust, the identity of S1 also flows to S2 simultaneously and S2 validates the identity by checking the trustedPrincipalList list to verify that it is a valid server principal. S2 also authenticates S1. The following steps take you through the configuration of C, S1, and S2.

Configuring client, C

Client C requires message layer authentication with a Secure Sockets Layer (SSL) transport. To accomplish this task:

1. Point the client to the sas.client.props file.

   Use the com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props property. All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required:
   com.ibm.CSI.performTransportAssocSSLTLSSupported=true,
   com.ibm.CSI.performTransportAssocSSLTLSRequired=false
3. Enable client authentication at the message layer.

   In this case, client authentication is supported but not required:
   com.ibm.CSI.performClientAuthenticationRequired=false,
   com.ibm.CSI.performClientAuthenticationSupported=true
4. Use all of the remaining defaults in the sas.client.props file.

Configuring server, S1

In the administrative console, server S1 is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. Server S1 is configured for outgoing requests to support identity assertion.

1. Configure S1 for incoming connections.
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.
2. Configure S1 for outgoing connections.
   a. Enable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.

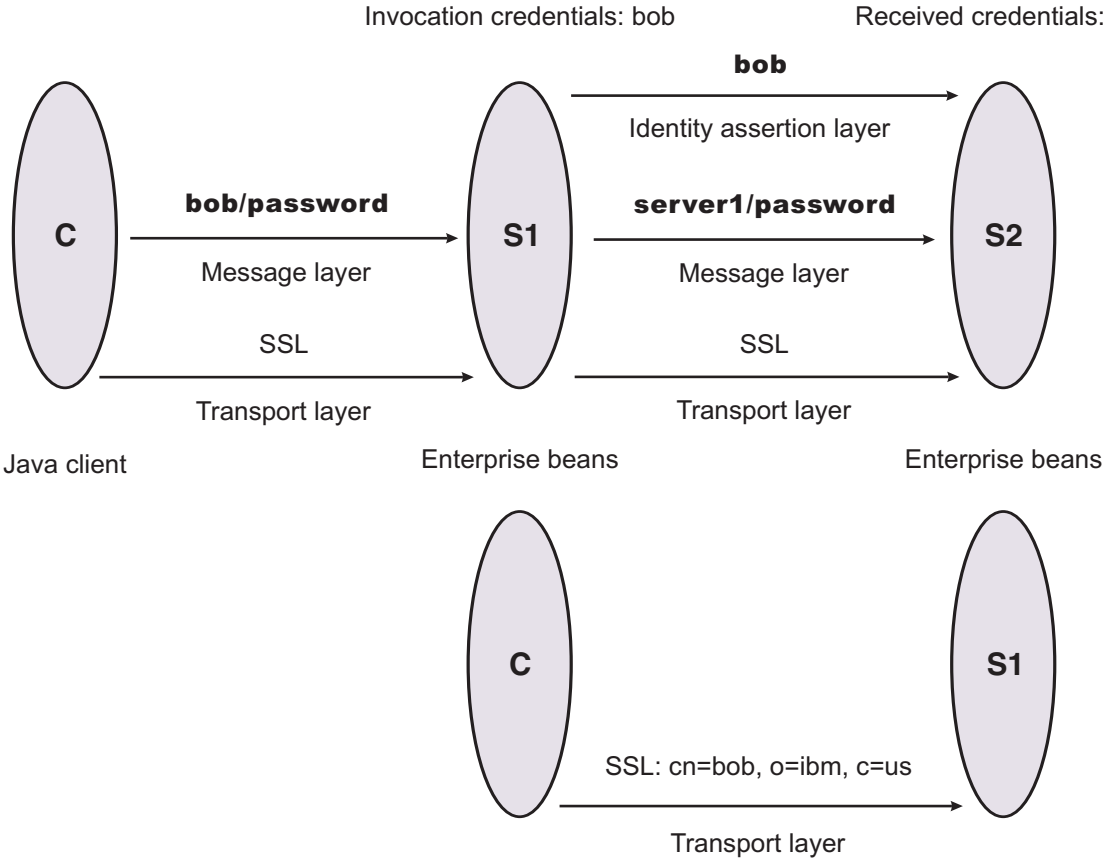
Configuring server, S2

In the administrative console, server S2 is configured for incoming requests to support identity assertion and to accept SSL connections. Complete the following steps to configure incoming connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

Scenario 2: Basic authentication, identity assertion, and client certificates:

This scenario is the same as Scenario 1, except for the interaction from client C2 to server S2. Therefore, the configuration of Scenario 1 still is valid, but you have to modify server S2 slightly and add a configuration for client C2. The configuration is not modified for C1 or S1.



Configuring client C2

Client C2 requires transport layer authentication (Secure Sockets Layer (SSL) client certificates). To configure transport layer authentication:

1. Point the client to the sas.client.props file.

   Use the com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props property. All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required:

   com.ibm.CSI.performTransportAssocSSLTLSSupported=true,
   com.ibm.CSI.performTransportAssocSSLTLSRequired=false
3. Disable client authentication at the message layer.

   com.ibm.CSI.performClientAuthenticationRequired=false,
   com.ibm.CSI.performClientAuthenticationSupported=false
4. Enable client authentication at the transport layer where it is supported, but not required:

   com.ibm.CSI.performTLClientAuthenticationRequired=false,
   com.ibm.CSI.performTLClientAuthenticationSupported=true

Configuring server, S2

In the administrative console, server S2 is configured for incoming requests to SSL client authentication and identity assertion. Configuration for outgoing requests is not relevant for this scenario.

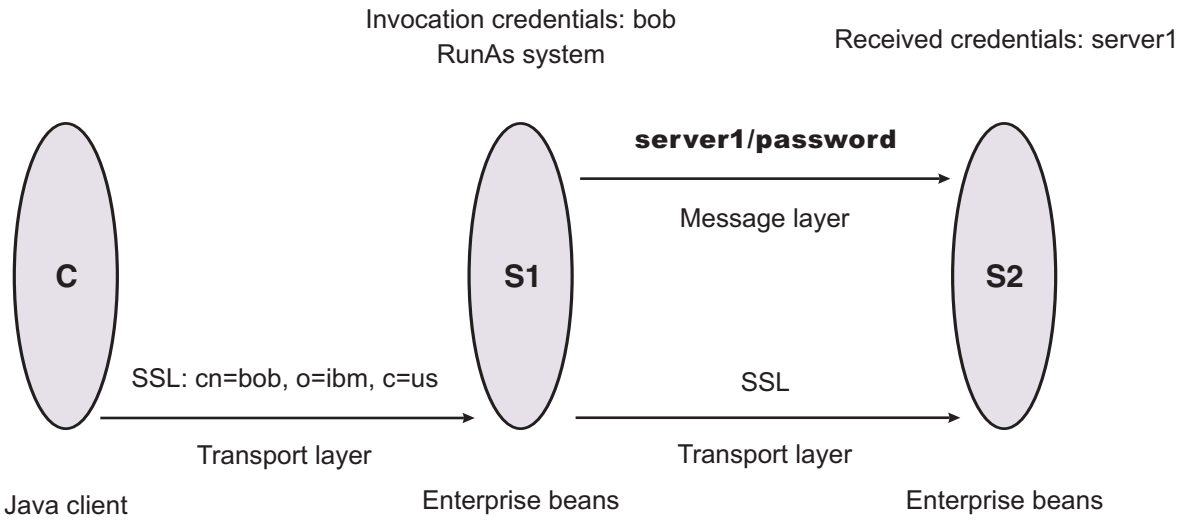
1. Enable identity assertion.

2. Disable user ID and password authentication.
3. Enable SSL.
4. Enable SSL client authentication.

You can mix and match these configuration options. However, a precedence exists as to which authentication features become the identity in the received credential:
1. Identity assertion
2. Message-layer client authentication (basic authentication or token)
3. Transport-layer client authentication (SSL certificates)

Scenario 3: Client certificate authentication and RunAs system:

This example presents a pure Java client, C, accessing a secure enterprise bean on S1.



C authenticates to S1 using Secure Sockets Layer (SSL) client certificates. S1 maps the common name of the distinguished name (DN) in the certificate to a user in the local registry. The user in this case is bob. The enterprise bean code on S1 accesses another enterprise bean on S2. Because the RunAs mode is system, the invocation credential is set as server1 for any outbound requests.

Configuring C

C requires transport layer authentication (SSL client certificates):
1. Point the client to the sas.client.props file.

   Use the com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props property. All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required:
   com.ibm.CSI.performTransportAssocSSLTLSSupported=true,
   com.ibm.CSI.performTransportAssocSSLTLSRequired=false
3. Disable client authentication at the message layer:com.ibm.CSI.performClientAuthenticationRequired=false,
   com.ibm.CSI.performClientAuthenticationSupported=false
4. Enable client authentication at the transport layer. It is supported, but not required:
   com.ibm.CSI.performTLClientAuthenticationRequired=false,
   com.ibm.CSI.performTLClientAuthenticationSupported=true

Configuring S1

In the administrative console, S1 is configured for incoming connections to support SSL with client certificate authentication. The S1 server is configured for outgoing requests to support message layer client authentication.

1. Configure S1 for incoming connections:
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.
2. Configure S1 for outgoing connections:
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.

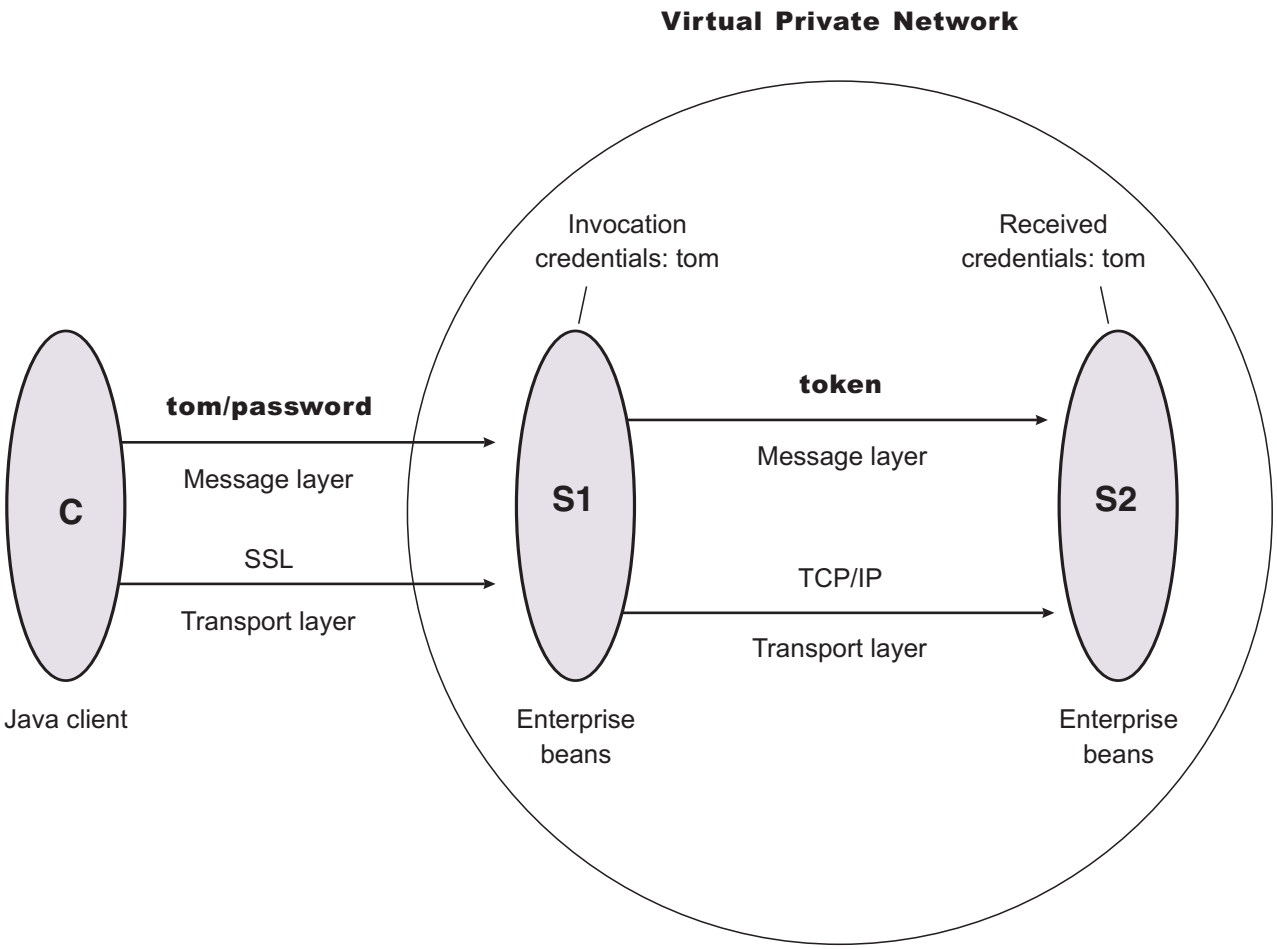
Configuring S2

In the administrative console, the S2 server is configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario.

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

Scenario 4: TCP/IP transport using a virtual private network:

This scenario illustrates the ability to choose TCP/IP as the transport when it is appropriate. In some cases, when two servers are on the same virtual private network (VPN), it can be appropriate to select TCP/IP as the transport for performance reasons because the VPN already encrypts the message.



Configuring C

C requires message layer authentication with an Secure Sockets Layer (SSL) transport:

1. Point the client to the sas.client.props file.

   Use the com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props property. All further configuration involves setting properties within this file.
2. Enable SSL. In this case, SSL is supported but not required: com.ibm.CSI.performTransportAssocSSLTLSSupported=true, com.ibm.CSI.performTransportAssocSSLTLSRequired=false
3. Enable client authentication at the message layer. In this case, client authentication is supported but not required: com.ibm.CSI.performClientAuthenticationRequired=false, com.ibm.CSI.performClientAuthenticationSupported=true
4. Use the remaining defaults in the sas.client.props file.

Configuring the S1 server

In the administrative console, the S1 server is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. The S1 server is configured for outgoing requests to support identity assertion.

1. Configure S1 for incoming connections:
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Enable SSL.

d. Disable SSL client certificate authentication.

2. Configure S1 for outgoing connections:
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Disable SSL.

It is possible to enable SSL for inbound connections and disable SSL for outbound connections. The same is true in reverse.

Configuring the S2 server

In the administrative console, the S2 server is configured for incoming requests to support identity assertion and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Disable SSL.

Configuring RMI over IIOP

Complete the following steps to configure Common Secure Interoperability Version 2 (CSIV2) and Security Authentication Service (SAS).

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

1. Determine how to configure security inbound and outbound at each point in your infrastructure.

   For example, you might have a Java client communicating with an Enterprise JavaBeans (EJB) application server, which in turn communicates to a downstream EJB application server.

   The Java client utilizes the `sas.client.props` file to configure outbound security. Pure clients must configure outbound security only.

   The upstream EJB application server configures inbound security to handle the correct type of authentication from the Java client. The upstream EJB application server utilizes the outbound security configuration when going to the downstream EJB application server.

   This type of authentication might be different than what you expect from the Java client into the upstream EJB application server. Security might be tighter between the pure client and the first EJB server, depending on your infrastructure. The downstream EJB server utilizes the inbound security configuration to accept requests from the upstream EJB server. These two servers require similar configuration options as well. If the downstream EJB application server communicates to other downstream servers, the outbound security might require a special configuration.

2. Specify the type of authentication.

   By default, authentication by a user ID and password is performed.

   Both Java client certificate authentication and identity assertion are disabled by default. If you want this type of authentication performed at every tier, use the CSIv2 authentication protocol configuration as is. However, if you have any special requirements where some servers authenticate differently from other servers, consider how to configure CSIv2 to its best advantage.

3. Configure clients and servers.

   Configuring a pure Java client is done through the `sas.client.props` file, where properties are modified.

   Configuring servers is always done from the administrative console or scripting, either from the security navigation for cell-level configurations or from the server security of the application server for

server-level configurations. If you want some servers to authenticate differently from others, modify some of the server-level configurations. When you modify the server-level configurations, you are overriding the cell-level configurations.

Configuring inbound transports

By using this configuration, you can configure a different transport for inbound security versus outbound security.

V6.0.x *Inbound transports* refer to the types of listener ports and their attributes that are opened to receive requests for this server. Both Common Secure Interoperability Specification, Version 2 (CSIv2) and Secure Authentication Service (SAS) have the ability to configure the transport.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

However, the following differences between the two protocols exist: V6.0.x

- CSIv2 is much more flexible than SAS, which requires Secure Sockets Layer (SSL); CSIv2 does not require SSL.
- SAS does not support SSL client certificate authentication, while CSIv2 does.
- CSIv2 can require SSL connections, while SAS only supports SSL connections.
- SAS always has two listener ports open: TCP/IP and SSL.
- CSIv2 can have as few as one listener port and as many as three listener ports. You can open one port for just TCP/IP or when SSL is required. You can open two ports when SSL is supported, and open three ports when SSL and SSL client certificate authentication is supported.

Complete the following steps to configure the Inbound transport panels in the administrative console:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under RMI/IIOP security, click **CSIv2 inbound transport** to select the type of transport and the SSL settings. By selecting the type of transport, as noted previously, you choose which listener ports you want to open. In addition, you disable the SSL client certificate authentication feature if you choose TCP/IP as the transport.
3. Select the SSL settings that correspond to an SSL transport. You can define these settings by accessing the SSL configurations panel. For more information, see tsec_sslconfiguration.dita.
4. Click **Apply** in the CSIv2 inbound transport panel.
5. Consider fixing the listener ports that you configured.

   You complete this action in a different panel, but think about this action now. Most endpoints are managed at a single location, which is why they do not display in the Inbound transport panels. Managing end points at a single location helps you decrease the number of conflicts in your configuration when you assign the endpoints. The location for SSL end points is at each server. The following port names are defined in the End points panel and are used for Object Request Broker (ORB) security:
   - CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS - CSIv2 Client Authentication SSL Port
   - CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS - CSIv2 SSL Port
   - V6.0.x SAS_SSL_SERVERAUTH_LISTENER_ADDRESS - SAS SSL Port
   - ORB_LISTENER_PORT - TCP/IP Port

   For an application server, click **Servers > Application servers >** *server_name*. Under Communications, click **Ports**. The Ports panel is displayed for the specified server.

   The Object Request Broker (ORB) on WebSphere Application Server uses a listener port for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) communications, and is statically specified using configuration dialogs or during migration. If you are working with a firewall, you must

specify a static port for the ORB listener and open that port on the firewall so that communication can pass through the specified port. The endPoint property for setting the ORB listener port is: ORB_LISTENER_ADDRESS.

Complete the following steps using the administrative console to specify the ORB_LISTENER_ADDRESS port or ports.

a. Click **Servers > Application Servers >** *server_name*. Under Communications, click **Ports > New**.
b. Select **ORB_LISTENER_ADDRESS** from the **Port name** field in the Configuration panel.
c. Enter the IP address, the fully qualified Domain Name System (DNS) host name, or the DNS host name by itself in the **Host** field. For example, if the host name is `myhost`, the fully qualified DNS name can be `myhost.myco.com` and the IP address can be `155.123.88.201`.
d. Enter the port number in the **Port** field. The port number specifies the port for which the service is configured to accept client requests. The port value is used with the host name. Using the previous example, the port number might be 9000.

6. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 inbound transport** to select the SSL settings that are used for inbound requests from CSIv2 clients. Remember that the CSIv2 protocol is used to inter-operate with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files need the right information for inter-operating with previous releases of WebSphere Application Server. For example, a previous release has a different truststore file than the Version 6 release. If you use the Version 6 keystore file, add the signer to the truststore file of the previous release for those clients connecting to this server.

The inbound transport configuration is complete. With this configuration, you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server that is used by users, the security configuration might be more secure. When requests go to back-end enterprise bean servers, you might lessen the security for performance reasons when you go outbound. With this flexibility you can design the right transport infrastructure to meet your needs.

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers:

1. Click **Save** in the administrative console to save any modifications to the configuration.
2. Stop and restart all servers, when synchronized.

Common Secure Interoperability Version 2 transport inbound settings:

Use this page to specify which listener ports to open and which Secure Sockets Layer (SSL) settings to use. These specifications determine which transport a client or upstream server uses to communicate with this server for incoming requests.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RMI/IIOP security > CSIv2 inbound transport**.

Transport:

Specifies whether client processes connect to the server using one of its connected transports.

You can choose to use either Secure Sockets Layer (SSL), TCP/IP or both as the inbound transport that a server supports. If you specify TCP/IP, the server only supports TCP/IP and cannot accept SSL connections. If you specify SSL-supported, this server can support either TCP/IP or SSL connections. If you specify SSL-required, then any server communicating with this one must use SSL.

If you specify SSL-supported or SSL-required, decide which set of SSL configuration settings you want to use for the inbound configuration. This decision determines which key file and trust file are used for inbound connections to this server.

TCP/IP
If you select **TCP/IP**, then the server opens a TCP/IP listener port only and all inbound requests do not have SSL protection.

SSL-required
If you select **SSL-required**, then the server opens an SSL listener port only and all inbound requests are received using SSL.

Important: V6.0.x If you set the active authentication protocol to **CSI and SAS**, then the server opens a TCP/IP listener port for the Secure Authentication Service (SAS) protocol regardless of this setting.

V6.0.x Only an SSL listener port is opened, and all requests come through SSL connections. If you choose **SSL-required**, you must also choose **CSI** as the active authentication protocol. If you choose **CSI and SAS**, SAS requires an open TCP/IP socket for some special requests.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

SSL-supported
If you select **SSL-supported**, then the server opens both a TCP/IP and an SSL listener port and most inbound requests are received using SSL.

V6.0.x By default, SSL ports for Common Secure Interoperability Version 2 (CSIv2) and Security Authentication Service (SAS) are dynamically generated. In cases where you need to fix the SSL ports on application servers, click **Servers** > **Application Servers** > *server_name*. Under Additional properties, click **Endpoint listeners**.

Provide a fixed port number for the following ports. A zero port number indicates that a dynamic assignment is made at runtime.

CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS

Default: SSL-Supported
Range: TCP/IP, SSL Required, SSL-Supported

SSL settings:

Specifies a list of predefined SSL settings to choose from for inbound connections.

Note: V6.0.x This option is available for non-z/OS platform servers when there is a version 6.0.x server in your environment. However, if your environment contains only Version 6.1 servers, this option does not apply.

These settings are configured at the SSL Repertoire panel. To access the SSL Repertoire panel, complete the following steps:

1. Clicking **Security > SSL certificate and key management**.
2. Under configuration settings, click **Manage endpoint security configurations and trust zones**.

3. Expand Inbound and click *inbound_configuration*.
4. Under Related items, click **SSL configurations**.

Data type:	String
Default:	DefaultSSLSettings
DefaultIIOPSSL	
Range:	Any SSL settings configured in the SSL Configuration Repertoire

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Default:	Enabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

z/OS SSL settings:

Specifies a list of predefined Secure Sockets Layer (SSL) settings for inbound connections. Configure these settings on the SSL panel by clicking Secure communications on the administrative console.

Secure Authentication Service inbound transport settings:

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. The SAS protocol is used to communicate securely to enterprise beans with previous releases of the application server.

To view this administrative console page, complete the following steps:
1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand RMI/IIOP security and click **SAS inbound transport**.

Attention: V6.0.x The panel associated with this article displays only when you have a Version 6.0.x server in your environment. SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

SSL Settings:

Specifies a list of predefined SSL settings to choose from for inbound connections.

These settings are configured on the Secure Sockets Layer (SSL) configuration panel. To access the SSL configuration panel, complete the following steps:
1. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
2. Expand Inbound > *configuration_name*.

3. Under Related Items, click **SSL configurations**.

Data type: String
Default: DefaultSSLSettings

Configuring outbound transports

By using this configuration, you can configure a different transport for inbound security versus outbound security.

Outbound transports refers to the transport that is used to connect to a downstream server. When you configure the outbound transport, consider the transports that the downstream servers support. If you are considering Secure Sockets Layer (SSL), also consider including the signers of the downstream servers in this server truststore file for the handshake to succeed.

When you select an SSL configuration, that configuration points to keystore and truststore files that contain the necessary signers.

If you configured client certificate authentication for this server by completing the following steps, then the downstream servers contain the signer certificate belonging to the server personal certificate:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under RMI/IIOP security, click **CSIv2 outbound authentication**

Complete the following steps to configure the outbound transport panels.

1. Select the type of transport and the SSL settings by clicking **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSIv2 outbound transport**. By selecting the type of transport, you choose the transport to use when connecting to downstream servers. The downstream servers support the transport that you choose. If you choose **SSL-Supported**, the transport that is used is negotiated during the connection. If both the client and server support SSL, always select the **SSL-Supported** option unless the request is considered a special request that does not require SSL, such as if an object request broker (ORB) is a request.
2. Select the **SSL required** option if you want to use Secure Sockets Layer communications with the outbound transport.

   If you select the **SSL required** option, you can select either the **Centrally managed** or **Use specific SSL alias** option.

   Centrally managed
   Enables you to specify an SSL configuration for particular scope such as the cell, node, server, or cluster in one location. To use the **Centrally managed** option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an outbound transport, you can override the inherited SSL configuration by specifying an SSL configuration for a particular endpoint. To specify an SSL configuration for an outbound transport, click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones** and expand **Outbound**.

   Use specific SSL alias
   Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu below the option.

   This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the steps described in “Creating a Secure Sockets Layer configuration” on page 1429.

3. V6.0.x Select the SSL that are settings used for outbound requests to downstream Secure Authentication Service (SAS) servers. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **SAS outbound transport**. Remember that the SAS protocol allows interoperability with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files have the correct information for inter-operating with previous releases of WebSphere Application Server. For example, a previous release has a different personal certificate than the Version 6.*x* release. If you use the keystore file from the Version 6.x release, you must add the signer to the truststore file of the previous release. Also, you must extract the signer for the Version 6.*x* release and import that signer into the truststore file of the previous release.

   Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

The outbound transport configuration is complete. With this configuration, you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server used by end users, the security configuration might be more secure. When requests go to back-end enterprise beans servers, you might consider less security for performance reasons when you go outbound. With this flexibility you can design a transport infrastructure that meets your needs.

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers.
- Click **Save** in the administrative console to save any modifications to the configuration.
- Stop and restart all servers, after synchronization.

Common Secure Interoperability Version 2 outbound transport settings:

Use this page to specify which transports and Secure Sockets Layer (SSL) settings this server uses when communicating with downstream servers for outbound requests.

To view this administrative console page, complete the following steps:
1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RMI/IIOP security > CSIv2 outbound transport**.

You also can view this administrative console by completing the following steps:
1. Click **Servers > Application servers > *server_name***.
2. Under Security, click **Server security**.
3. Under Additional properties, click **CSIv2 outbound transport**.

Transport:

Specifies whether the client processes connect to the server using one of the server-connected transports.

You can choose to use either SSL, TCP/IP, or Both as the outbound transport that a server supports. If you specify `TCP/IP`, the server supports only TCP/IP and cannot initiate SSL connections with downstream servers. If you specify `SSL-supported`, this server can initiate either TCP/IP or SSL connections. If you specify `SSL-required`, this server must use SSL to initiate connections to downstream servers. When you do specify SSL, decide which set of SSL configuration settings you want to use for the outbound configuration.

This decision determines which keyfile and trustfile to use for outbound connections to downstream servers.

Consider the following options:

TCP/IP
If you select this option, the server opens TCP/IP connections with downstream servers only.

SSL-required
If you select this option, the server opens SSL connections with downstream servers.

SSL-supported
If you select this option, the server opens SSL connections with any downstream server that supports them and opens TCP/IP connections with any downstream servers that do not support SSL.

Default:	SSL-supported
Range:	TCP/IP, SSL-required, SSL-supported

SSL settings:

Specifies a list of predefined SSL settings for outbound connections. These settings are configured at the SSL Configuration Repertoires panel.

To access the panel, complete the following steps:

1. Click **Security > SSL certificate and key management**.
2. Under Configuration settings, click **Manage endpoint security configurations and trust zones**.
3. Expand Outbound > *outbound_configuration_name*.
4. Under Related items, click **SSL configurations**.

Data type:	String
Range:	Any SSL settings that are configured in the SSL Configuration Repertoires panel

Note: V6.0.x This field is available only if a Version 6.0.x server exists in your environment.

SSL enabled:

Specifies whether secure socket communication is enabled to the server.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Default:	Enabled

Use specific SSL alias:

Specifies the SSL configuration alias that you want to use for outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI (LDAP) protocol.

Secure Authentication Service outbound transport settings:

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand RMI/IIOP security and click **SAS outbound transport**.

Attention: V6.0.x The panel associated with this article displays only when you have a Version 6.0.*x* server in your environment.

SSL settings:

Specifies a list of predefined Secure Sockets Layer (SSL) settings to choose from for outbound connections.

These settings are configured on the SSL configuration panel. To access the SSL configuration panel, complete the following steps:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
2. Expand Outbound > *configuration_name*.
3. Under Related Items, click **SSL configurations**.

Data type:	String
Default:	DefaultSSLSettings

Performing identity mapping for authorization across servers in different realms

Identity mapping is a one-to-one mapping of a user identity between two servers so that the proper authorization decisions are made by downstream servers. Identity mapping is necessary when the integration of servers is needed, but the user registries are different and not shared between the systems.

The following topics are covered in this section:

- Configuring inbound identity mapping
- Configuring outbound identity mapping to a different target realm

Configuring inbound identity mapping:

For inbound identity mapping, write a custom login module and configure WebSphere Application Server to run the login module first within the system login configurations. Consider the following steps when you write your custom login module.

1. Get the inbound user identity from the callbacks and map the identity, if necessary This step occurs in the login method of the login module. A valid authentication has either or both NameCallback and the WSCredTokenCallback callbacks present. The following code sample shows you how to determine the user identity:

```
javax.security.auth.callback.Callback callbacks[] =
   new javax.security.auth.callback.Callback[3];
 callbacks[0] = new javax.security.auth.callback.NameCallback("");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback
     ("Password: ", false);
 callbacks[2] = new com.ibm.websphere.security.auth.callback.
     WSCredTokenCallbackImpl("");
 callbacks[3] = new com.ibm.wsspi.security.auth.callback.
     WSTokenHolderCallback("");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
```

```
{
 // Handles exceptions
 throw new WSLoginFailedException (e.getMessage(), e);
}

// Shows which callbacks contain information
boolean identitySwitched = false;
String uid = ((NameCallback) callbacks[0]).getName();
char password[] = ((PasswordCallback) callbacks[1]).getPassword();
byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
java.util.List authzTokenList = ((WSTokenHolderCallback)
    callbacks[3]).getTokenHolderList();

if (credToken != null)
{
 try
 {
  String uniqueID = WSSecurityPropagationHelper.validateLTPAToken(credToken);
  String realm = WSSecurityPropagationHelper.getRealmFromUniqueID (uniqueID);
      // Now set the string to the UID so that you can use the result for either
      // mapping or logging in.
  uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
 }
 catch (Exception e)
 {
  // Handles the exception
 }
}
else if (uid == null)
{
    // Throws an exception if authentication data is not valid.
    // You must have either UID or CredToken
 throw new WSLoginFailedException("invalid authentication data.");
}
else if (uid != null && password != null)
{
    // This is a typical authentication. You can choose to map this ID to
    // another ID or you can skip it and allow WebSphere Application Server
    // to log in for you. When passwords are presented, be very careful to not
    // validate the password because this is the initial authentication.

 return true;
}

   // If desired, map this uid to something else and set the identitySwitched
   // boolean. If the identity was changed, clear the propagated attributes
   // below so they are not used incorrectly.
uid = myCustomMappingRoutine (uid);

   // Clear the propagated attributes because they are no longer applicable
   // to the new identity
if (identitySwitched)
{
 ((WSTokenHolderCallback) callbacks[3]).setTokenHolderList(null);
}
```

2. Check to see if attribute propagation occurred and if the attributes for the user are already present when the identity remains the same. Check to see if the user attributes are already present from the sending server to avoid duplicate calls to the user registry lookup. To check for the user attributes, use a method on the WSTokenHolderCallback callback that analyzes the information present in the callback to determine if the information is sufficient for WebSphere Application Server to create a Subject. The following code sample checks for the user attributes:

```
boolean requiresLogin =
((com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback)
callbacks[2]).requiresLogin();
```

If sufficient attributes are not present to form the WSCredential and the WSPrincipal objects that are needed to perform authorization, the previous code sample returns a `true` result. When the result is `false`, you can choose to discontinue processing as the necessary information exists to create the Subject without performing additional remote user registry calls.

3. **Optional:** Look up the required attributes from the user registry, put the attributes in a hashtable, and add the hashtable to the shared state. If the identity is switched in this login module, you must complete the following steps:
   a. Create the hashtable of attributes, as shown in the following example.
   b. Add the hashtable to the shared state.

   If the identity is not switched, but the value of the requiresLogin code sample shown previously is `true`, you can create the hashtable of attributes. However, you are not required to create a hashtable in this situation as WebSphere Application Server handles the login for you. However, you might consider creating a hashtable to gather attributes in special cases where you are using your own special user registry. Creating a UserRegistry implementation, using a hashtable, and letting WebSphere Application Server gather the user attributes for you might be the easiest solution. The following table shows how to create a hashtable of user attributes:

```
if (requiresLogin || identitySwitched)
 {
  // Retrieves the default InitialContext for this server.
  javax.naming.InitialContext ctx = new javax.naming.InitialContext();

  // Retrieves the local UserRegistry implementation.
  com.ibm.websphere.security.UserRegistry reg = (com.ibm.websphere.
        security.UserRegistry)
  ctx.lookup("UserRegistry");

     // Retrieves the user registry uniqueID based on the uid specified
     // in the NameCallback.
  String uniqueid = reg.getUniqueUserId(uid);
   uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);

     // Retrieves the display name from the user registry based on the uniqueID.
  String securityName = reg.getUserSecurityName(uid);

     // Retrieves the groups associated with the uniqueID.
  java.util.List groupList = reg.getUniqueGroupIds(uid);

     // Creates the java.util.Hashtable with the information that you gathered
     // from the UserRegistry implementation.
  java.util.Hashtable hashtable = new java.util.Hashtable();
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_UNIQUEID, uniqueid);
     hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_SECURITYNAME, securityName);
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_GROUPS, groupList);

     // Adds a cache key that is used as part of the lookup mechanism for
     // the created Subject. The cache key can be an object, but should have
     // an implemented toString method. Make sure that the cacheKey contains
     // enough information to scope it to the user and any additional attributes
     // that you are using. If you do not specify this property the Subject is
     // scoped to the returned WSCREDENTIAL_UNIQUEID, by default.
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_CACHE_KEY, "myCustomAttribute" + uniqueid);
  // Adds the hashtable to the sharedState of the Subject.
  _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_PROPERTIES_KEY, hashtable);
 }
```

The following rules define in more detail how a hashtable login is performed. You must use a java.util.Hashtable object in either the Subject (public or private credential set) or the shared-state

HashMap. The com.ibm.wsspi.security.token.AttributeNameConstants class defines the keys that contain the user information. If the Hashtable object is put into the shared state of the login context using a custom login module that is listed prior to the Lightweight Third Party Authentication (LTPA) login module, the value of the java.util.Hashtable object is searched using the following key within the shared-state hashMap:

Property
com.ibm.wsspi.security.cred.propertiesObject

Reference to the property
AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY

Explanation
This key searches for the Hashtable object that contains the required properties in the shared state of the login context.

Expected result
A java.util.Hashtable object.

If a java.util.Hashtable object is found either inside the Subject or within the shared state area, verify that the following properties are present in the hashtable:

Property
com.ibm.wsspi.security.cred.uniqueId

Reference to the property
AttributeNameConstants.WSCREDENTIAL_UNIQUEID

Returns
java.util.String

Explanation
The value of the property must be a unique representation of the user. For the WebSphere Application Server default implementation, this property represents the information that is stored in the application authorization table. The information is located in the application deployment descriptor after it is deployed and user-to-role mapping is performed. See the expected format examples if the user to role mapping is performed using a lookup to a WebSphere Application Server user registry implementation.

If a third-party authorization provider overrides the user-to-role mapping, then the third-party authorization provider defines the format. To ensure compatibility with the WebSphere Application Server default implementation for the unique ID value, call the WebSphere Application Server public String getUniqueUserId(String userSecurityName) UserRegistry method.

Expected format examples

Realm	Format (uniqueUserId)
Lightweight Directory Access Protocol (LDAP)	`ldaphost.austin.ibm.com:389/cn=user,o=ibm,c=us`
Windows	`MYWINHOST/S-1-5-21-963918322-163748893-4247568029-500`
UNIX	`MYUNIXHOST/32`

The com.ibm.wsspi.security.cred.uniqueId property is required.

Property
com.ibm.wsspi.security.cred.securityName

Reference to the property
AttributeNameConstants. WSCREDENTIAL_ SECURITYNAME

Returns

java.util.String

Explanation

This property searches for the securityName of the authentication user. This name is commonly called the *display name* or *short name*. WebSphere Application Server uses the securityName attribute for the getRemoteUser, getUserPrincipal and getCallerPrincipal application programming interfaces (APIs). To ensure compatibility with the WebSphere Application Server default implementation for the securityName value, call the WebSphere Application Server public String getUserSecurityName(String uniqueUserId) UserRegistry method.

Expected format examples

Realm	Format (uniqueUserId)
LDAP	user (*LDAP UID*)
Windows	user (*Windows username*)
UNIX	user (*UNIX username*)

The com.ibm.wsspi.security.cred.securityName property is required.

Property

com.ibm.wsspi.security.cred.groups

Reference to the property

AttributeNameConstants. WSCREDENTIAL_GROUPS

Returns

java.util.ArrayList

Explanation

This key searches for the array list of groups to which the user belongs. The groups are specified in the *realm_name*/*user_name* format. The format of these groups is important as the groups are used by the WebSphere Application Server authorization engine for group-to-role mappings in the deployment descriptor. The format that is provided must match the format expected by the WebSphere Application Server default implementation. When you use a third-party authorization provider, you must use the format that is expected by the third-party provider. To ensure compatibility with the WebSphere Application Server default implementation for the unique group IDs value, call the WebSphere Application Server public List getUniqueGroupIds(String uniqueUserId) UserRegistry method.

Expected format examples for each group in the array list

Realm	Format
LDAP	`ldap1.austin.ibm.com:389/cn=group1,o=ibm,c=us`
Windows	`MYWINREALM/S-1-5-32-544`
UNIX	`MY/S-1-5-32-544`

The com.ibm.wsspi.security.cred.groups property is not required. A user is not required to have associated groups.

Property

com.ibm.wsspi.security.cred.cacheKey

Reference to the property

AttributeNameConstants. WSCREDENTIAL_CACHE_KEY

Returns

java.lang.Object

Explanation

This key property can specify an object that represents the unique properties of the login, including the user-specific information and the user dynamic attributes that might affect uniqueness. For example, when the user logs in from location A, which might affect their access control, the cache key needs to include location A so that the Subject that is received is the correct Subject for the current location.

This com.ibm.wsspi.security.cred.cacheKey property is not required. When this property is not specified, the cache lookup is the value that is specified for WSCREDENTIAL_UNIQUEID. When this information is found in the java.util.Hashtable object, WebSphere Application Server creates a Subject similar to the Subject that goes through the normal login process at least for LTPA. The new Subject contains a WSCredential object and a WSPrincipal object that is fully populated with the information found in the Hashtable object.

4. Add your custom login module into the RMI_INBOUND, WEB_INBOUND, and DEFAULT Java Authentication and Authorization Service (JAAS) system login configurations. Configure the RMI_INBOUND login configuration so that WebSphere Application Server loads your new custom login module first.
   a. Click **Security > Secure administration, applications, and infrastructure > Java Authentication and Authorization Service > System logins > RMI_INBOUND**
   b. Under Additional Properties, click **JAAS login modules > New** to add your login module to the RMI_INBOUND configuration.
   c. Return to the JAAS login modules panel for RMI_INBOUND.
   d. Click **Set order** to change the order that the login modules are loaded so that WebSphere Application Server loads your custom login module first. Use the **Move Up** or **Move Down** buttons to arrange the order of the login modules.
   e. Repeat the previous three steps for the WEB_INBOUND and DEFAULT login configurations.

This process configures identity mapping for an inbound request.

The “Example: Custom login module for inbound mapping” on page 1316 topic shows a custom login module that creates a java.util.Hashtable hashtable that is based on the specified NameCallback callback. The java.util.Hashtable hashtable is added to the sharedState java.util.Map map so that the WebSphere Application Server login modules can locate the information in the hashtable.

Identity mapping:

Identity mapping is a one-to-one mapping of a user identity between two servers so that the proper authorization decisions are made by downstream servers. Identity mapping is necessary when the integration of servers is needed, but the user registries are different and not shared between the systems.

In most cases, requests flow downstream between two servers that are part of the same security domain. In WebSphere Application Server, two servers that are members of the same cell are also members of the same security domain. In the same cell, the two servers have the same user registry and the same Lightweight Third Party Authentication (LTPA) keys for token encryption. These two commonalities ensure that the LTPA token, among other user attributes, which flows between the two servers, not only can be decrypted and validated, but also the user identity in the token can be mapped to attributes that are recognized by the authorization engine.

The most reliable and recommended configuration involves two servers within the same cell. However, sometimes you need to integrate multiple systems that cannot use the same user registry. When the user registries are different between two servers, the security domain or realm of the target server does not match the security domain of the sending server.

WebSphere Application Server enables mapping to occur either before sending the request outbound or before enabling the existing security credentials to flow to the target server. The credentials are mapped inbound with the specification that the target realm is trusted.

An alternative to mapping is to send the user identity without the token or the password to a target server without actually mapping the identity. The use of the user identity is based on trust between the two servers. Use Common Secure Interoperability Version 2 (CSIv2) identity assertion. When enabled, the server sends just the X.509 certificate, principal name, or distinguished name (DN) based upon what was used by the original client to perform the initial authentication. During CSIv2 identity assertion, trust is established between WebSphere Application Servers.

The user identity must exist in the target user registry for identity assertion to work. This process can also enable interoperability between other Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 and higher compliant application servers. If both the sending server and target servers have identity assertion configured, WebSphere Application Server always uses this method of authentication, even when both servers are in the same security domain. For more information on CSIv2 identity assertion, see “Identity assertion” on page 1233.

When the user identity is not present in the user registry of the target server, identity mapping must occur either before the request is sent outbound or when the request comes inbound. This decision depends upon your environment and requirements. However, it is typically easier to map the user identity before the request is sent outbound for the following reasons:

- You know the user identity of the existing credential as it comes from the user registry of the sending server.
- You do not have to worry about sharing Lightweight Third Party Authentication (LTPA) keys with the other target realm because you are not mapping the identity to LTPA credentials. Typically, you are mapping the identity to a user ID and password that are present in the user registry of the target realm.

When you do perform outbound mapping, in most cases, it is recommended that you use Secure Sockets Layer (SSL) to protect the integrity and confidentiality of the security information sent across the network. If LTPA keys are not shared between servers, an LTPA token cannot be validated at the inbound server. In this case, outbound mapping is necessary because the user identity cannot be determined at the inbound server to do inbound mapping. For more information, see “Configuring outbound mapping to a different target realm” on page 1319.

When you need inbound mapping, potentially due to the mapping capabilities of the inbound server, you must ensure that both servers have the same LTPA keys so that you can get access to the user identity. Typically, in secure communications between servers, an LTPA token is passed into the WSCredTokenCallback callback of the inbound JAAS login configuration for the purposes of client authentication. A method is available that enables you to open the LTPA token, if valid, and get access to the user unique ID so that mapping can be performed. For more information, see “Configuring inbound identity mapping” on page 1310. In other cases, such as identity assertion, you might receive a user name in the NameCallback callback of the inbound login configuration that enables you to map the identity.

Example: Custom login module for inbound mapping:

This sample shows a custom login module that creates a java.util.Hashtable hashtable that is based on the specified NameCallback callback. The java.util.Hashtable hashtable is added to the sharedState java.util.Map map so that the WebSphere Application Server login modules can locate the information in the Hashtable.

```
public customLoginModule()
{

public void initialize(Subject subject, CallbackHandler callbackHandler,
   Map sharedState, Map options)
{
 // (For more information on initialization, see
```

```
 //  Custom login module development for a system login configuration.)
 _sharedState = sharedState;
}

public boolean login() throws LoginException
{
 // (For more information on what to do during login, see
 //  Custom login module development for a system login configuration.)

   // Handles the WSTokenHolderCallback to see if this is an initial or
   // propagation login.
 javax.security.auth.callback.Callback callbacks[] =
     new javax.security.auth.callback.Callback[3];
 callbacks[0] = new javax.security.auth.callback.NameCallback("");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback(
     "Password: ", false);
 callbacks[2] = new com.ibm.websphere.security.auth.callback.
     WSCredTokenCallbackImpl("");
 callbacks[3] = new com.ibm.wsspi.security.auth.callback.
     WSTokenHolderCallback("");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles the exception
 }

 // Determines which callbacks contain information
 boolean identitySwitched = false;
 String uid = ((NameCallback) callbacks[0]).getName();
 char password[] = ((PasswordCallback) callbacks[1]).getPassword();
 byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
 java.util.List authzTokenList = ((WSTokenHolderCallback) callbacks[3]).
     getTokenHolderList();

 if (credToken != null)
 {
  try
  {
   String uniqueID = WSSecurityPropagationHelper.validateLTPAToken(credToken);
   String realm = WSSecurityPropagationHelper.getRealmFromUniqueID (uniqueID);
       // Set the string to the UID so you can use the information to either
       // map or login.
   uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
  }
  catch (Exception e)
  {
   // handle exception
  }
 }
 else if (uid == null)
 {
  // The authentication data is not valid. You must have either UID
    // or CredToken
  throw new WSLoginFailedException("invalid authentication data.");
 }
 else if (uid != null && password != null)
 {
     // This is a typical authentication. You can choose to map this ID to
     // another ID or you can skip it and allow WebSphere Application Server
     // to log in for you. When passwords are presented, be very careful not
     // to validate the password because this is the initial authentication.

  return true;
```

```
}

 // You can map this uid to something else and set the identitySwitched
 // boolean. If the identity is changed, clear the following propagated
 // attributes so they are not used incorrectly.
uid = myCustomMappingRoutine (uid);

// Clear the propagated attributes because they no longer apply to the new identity
if (identitySwitched)
{
 ((WSTokenHolderCallback) callbacks[3]).setTokenHolderList(null);
}
boolean requiresLogin = ((com.ibm.wsspi.security.auth.callback.
    WSTokenHolderCallback) callbacks[2]).requiresLogin();

if (requiresLogin || identitySwitched)
{
 // Retrieves the default InitialContext for this server.
 javax.naming.InitialContext ctx = new javax.naming.InitialContext();

 // Retrieves the local UserRegistry object.
 com.ibm.websphere.security.UserRegistry reg =
        (com.ibm.websphere.security.UserRegistry) ctx.lookup("UserRegistry");

 // Retrieves the registry uniqueID based on the uid that is specified
    // in the NameCallback.
 String uniqueid = reg.getUniqueUserId(uid);
  uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);

 // Retrieves the display name from the user registry based on the uniqueID.
 String securityName = reg.getUserSecurityName(uid);

 // Retrieves the groups associated with this uniqueID.
 java.util.List groupList = reg.getUniqueGroupIds(uid);

 // Creates the java.util.Hashtable with the information that you gathered
    // from the UserRegistry.
 java.util.Hashtable hashtable = new java.util.Hashtable();
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_UNIQUEID, uniqueid);
    hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_SECURITYNAME, securityName);
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_GROUPS, groupList);

 // Adds a cache key that is used as part of the lookup mechanism for
 // the created Subject. The cache key can be an object, but has
 // an implemented toString method. Make sure the cacheKey contains enough
 // information to scope it to the user and any additional attributes you are
    // using. If you do not specify this property, the Subject is scoped to the
 // WSCREDENTIAL_UNIQUEID returned, by default.
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_CACHE_KEY, "myCustomAttribute" + uniqueid);
 // Adds the hashtable to the shared state of the Subject.
 _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_PROPERTIES_KEY, hashtable);
}
else if (requiresLogin == false)
{
 // For more information on this section, see
    // "Security attribute propagation" on page 1206.
 // If you added a custom Token implementation, you can search through the
    // token holder list for it to deserialize.
 // Note: Any Java objects are automatically deserialized by
    // wsMapDefaultInboundLoginModule

 for (int i=0; i<authzTokenList.size(); i++)
```

```
  {
   if (authzTokenList[i].getName().equals("com.acme.MyCustomTokenImpl")
   {
    byte[] myTokenBytes = authzTokenList[i].getBytes();

          // Passes these bytes into the constructor of your implementation
          // class for deserialization.
    com.acme.MyCustomTokenImpl myTokenImpl =
          new com.acme.MyCustomTokenImpl(myTokenBytes);
   }
  }
 }
}

public boolean commit() throws LoginException
{
 // (For more information on what to do during a commit, see
  //   Custom login module development for a system login configuration.)
}

// Defines your login module variables
com.ibm.wsspi.security.token.AuthorizationToken customAuthzToken = null;
com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
java.util.Map _sharedState = null;
}
```

Configuring outbound mapping to a different target realm:

By default, when WebSphere Application Server makes an outbound request from one server to another server in a different security realm, the request is rejected. This topic details alternatives for enabling one server to send outbound requests to a target server in a different realm.

This outbound request is rejected to protect against a rogue server reading potentially sensitive information if successfully impersonating the home of the object. Select one of the following alternative procedures so that one server can send outbound requests to a target server in a different realm. When you are finished with a procedure on the administrative console, click **Apply**.

- Do not perform mapping. Instead, allow the existing security information to flow to a trusted target server, even if the target server resides in a different realm. Complete the following steps in the administrative console:
  1. Click **Security > Secure administration, applications, and infrastructure**.
  2. Under RMI/IIOP security, click **CSIv2 outbound authentication**.
  3. Specify the target realms in the **Trusted target realms** field. You can specify each trusted target realm that is separated by a pipe (|) character. For example, specify *server_name*.*domain*:*port_number* for a Lightweight Directory Access Protocol (LDAP) server or the machine name for local operating system. If you want to propagate security attributes to a different target realm, you must specify that target realm in the **Trusted target realms** field.
- Use the Java Authentication and Authorization Service (JAAS) WSLogin application login configuration to create a basic authentication Subject that contains the credentials of the new target realm. This configuration enables you to log in with a realm, user ID, and password that are specific to the user registry of the target realm. You can provide the login information from within the Java 2 Platform, Enterprise Edition (J2EE) application that is making the outbound request or from within the RMI_OUTBOUND system login configuration. These two login options are described in the following information:
  1. Use the WSLogin application login configuration from within the J2EE application to log in and get a Subject that contains the user ID and the password of the target realm. The application can wrap the remote call with a WSSubject.doAs call. For an example, see “Example: Using the WSLogin configuration to create a basic authentication subject” on page 1320.

2. Use the code sample in “Example: Using the WSLogin configuration to create a basic authentication subject” from this plug point within the RMI_OUTBOUND login configuration. Every outbound Remote Method Invocation (RMI) request passes through this login configuration when it is enabled. Complete the following steps to enable and plug in this login configuration:
   a. Click **Security > Secure administration, applications, and infrastructure**.
   b. Under RMI/IIOP security, click **CSIv2 outbound authentication**.
   c. Select the **Custom outbound mapping** option. If the **Security Attribute Propagation** option is selected, then WebSphere Application Server is already using this login configuration and you do not need to enable custom outbound mapping.
   d. Write a custom login module. For more information, see Custom login module development for a system login configuration.

      The “Example: Sample login configuration for RMI_OUTBOUND” on page 1322 shows a custom login module that determines whether the realm names match. In this example, the realm names do not match so the WSLoginmodule is used to create a basic authentication Subject based on custom mapping rules. The custom mapping rules are specific to the customer environment and must be implemented using a realm to user ID and password mapping utility.
   e. Configure the RMI_OUTBOUND login configuration so that your new custom login module is first in the list.
      1) Click **Security > Secure administration, applications, and infrastructure**.
      2) Under Java Authentication and Authorization Service, click **System logins > RMI_OUTBOUND**
      3) Under Additional Properties, click **JAAS login modules > New** to add your login module to the RMI_OUTBOUND configuration.
      4) Return to the JAAS login modules panel for RMI_OUTBOUND.
      5) Click **Set order** to change the order that the login modules are loaded so that your custom login is loaded first.

- Add the use_realm_callback and use_appcontext_callback options to the outbound mapping module for WSLogin. To add these options, complete the following steps:
  1. Click **Security > Secure administration, applications, and infrastructure**.
  2. Under Java Authentication and Authorization Service, click **Application logins > WSLogin**.
  3. Under Additional properties, click **JAAS login modules > com.ibm.ws.security.common.auth.module.WSLoginModuleImpl**.
  4. Under Additional properties, click **Custom Properties > New**.
  5. On the Custom properties panel, enter `use_realm_callback` in the **Name** field and `true` in the **Value** field.
  6. Click **OK**.
  7. Click **New** to enter the second custom property.
  8. On the Custom properties panel, enter `use_appcontext_callback` in the **Name** field and `true` in the **Value** field.

  The following changes are made to the `security.xml` file:

```
<entries xmi:id="JAASConfigurationEntry_2" alias="WSLogin">
 <loginModules xmi:id="JAASLoginModule_2"
  moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"
  authenticationStrategy="REQUIRED">
  <options xmi:id="Property_2" name="delegate"
   value="com.ibm.ws.security.common.auth.module.WSLoginModuleImpl"/>
  <options xmi:id="Property_3" name="use_realm_callback" value="true"/>
  <options xmi:id="Property_4" name="use_appcontext_callback" value="true"/>
 </loginModules>
</entries>
```

Example: Using the WSLogin configuration to create a basic authentication subject:

This example shows how to use the WSLogin application login configuration from within a Java 2 Platform, Enterprise Edition (J2EE) application to log in and get a Subject that contains the user ID and the password of the target realm.

```
javax.security.auth.Subject subject = null;

try
{
  // Create a login context using the WSLogin login configuration and specify a
  // user ID, target realm, and password. Note: If the target_realm_name is the
  // same as the current realm, an authenticated Subject is created. However, if
  // the target_realm_name is different from the current realm, a basic
  // authentication Subject is created that is not validated. This unvalidated
  // Subject is created so that you can send a request to the different target
  // realm with valid security credentials for that realm.
  javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
   new WSCallbackHandlerImpl("userid", "target_realm_name", "password"));

  // Note: The following code is an alternative that validates the user ID and
  // password specified against the target realm. The code performs a remote call
  // to the target server and will return  true if the user ID and password are
  // valid and false if the user ID and password are not valid. If false is
  // returned, a WSLoginFailedException exception is created. You can catch
  // that exception and perform a retry or stop the request from flowing by
  // allowing that exception to surface out of this login.

  // ALTERNATIVE LOGIN CONTEXT THAT VALIDATES THE USER ID AND PASSWORD TO THE
  // TARGET REALM

  /****  currently remarked out ****
  java.util.Map appContext = new java.util.HashMap();
              appContext.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
                             "com.ibm.websphere.naming.WsnInitialContextFactory");
              appContext.put(javax.naming.Context.PROVIDER_URL,
                             "corbaloc:iiop:target_host:2809");

  javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
   new WSCallbackHandlerImpl("userid", "target_realm_name", "password", appContext));
  **** currently remarked out  ****/

  // Starts the login
  ctx.login();

  // Gets the Subject from the context
  subject = ctx.getSubject();
 }
 catch (javax.security.auth.login.LoginException e)
 {
  throw new com.ibm.websphere.security.auth.WSLoginFailedException (e.getMessage(), e);
 }

 if (subject != null)
 {
  // Defines a privileged action that encapsulates your remote request.
java.security.PrivilegedAction myAction = java.security.PrivilegedAction()
  {
   public Object run()
   {
    // Assumes a proxy is already defined. This example method returns a String
    return proxy.remoteRequest();
   }
  });

  // Starts this action using the basic authentication Subject needed for
```

```
    // the target realm security requirements.
  String myResult = (String) com.ibm.websphere.security.auth.WSSubject.doAs
        (subject, myAction);
 }
```

Example: Sample login configuration for RMI_OUTBOUND:

This example shows a sample login configuration for RMI_OUTBOUND that determines whether the realm names match between two servers.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
     // (For more information on what to do during initialization, see
     // Custom login module development for a system login configuration.)
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // Custom login module development for a system login configuration.)

  // Gets the WSProtocolPolicyCallback object
  Callback callbacks[] = new Callback[1];
   callbacks[0] = new com.ibm.wsspi.security.auth.callback.
          WSProtocolPolicyCallback("Protocol Policy Callback: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles the exception
  }

     // Receives the RMI (CSIv2) policy object for checking the target realm
     // based upon information from the IOR.
     // Note: This object can be used to perform additional security checks.
     // See the application programming interface (API) documentation for
     // more information.
  csiv2PerformPolicy = (CSIv2PerformPolicy) ((WSProtocolPolicyCallback)callbacks[0]).
        getProtocolPolicy();

  // Checks if the realms do not match. If they do not match, then log in to
     // perform a mapping
  if (!csiv2PerformPolicy.getTargetSecurityName().equalsIgnoreCase(csiv2PerformPolicy.
         getCurrentSecurityName()))
  {
   try
   {
    // Do some custom realm -> user ID and password mapping
    MyBasicAuthDataObject myBasicAuthData = MyMappingLogin.lookup
          (csiv2PerformPolicy.getTargetSecurityName());

          // Creates the login context with basic authentication data gathered from
          // custom mapping
     javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
      new WSCallbackHandlerImpl(myBasicAuthData.userid,
        csiv2PerformPolicy.getTargetSecurityName(),
                    myBasicAuthData.password));

     // Starts the login
     ctx.login();
```

```
            // Gets the Subject from the context. This subject is used to replace
            // the passed-in Subject during the commit phase.
    basic_auth_subject = ctx.getSubject();
   }
   catch (javax.security.auth.login.LoginException e)
   {
    throw new com.ibm.websphere.security.auth.
              WSLoginFailedException (e.getMessage(), e);
   }
 }
}

public boolean commit() throws LoginException
{
    // (For more information on what to do during commit, see
    // Custom login module development for a system login configuration.)

 if (basic_auth_subject != null)
 {
      // Removes everything from the current Subject and adds everything from the
      // basic_auth_subject
  try
  {
   public final Subject basic_auth_subject_priv = basic_auth_subject;
         // Do this in a doPrivileged code block so that application code
         // does not need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.
            PrivilegedExceptionAction()
   {
    public Object run() throws WSLoginFailedException
    {
              // Removes everything user-specific from the current outbound
              // Subject. This a temporary Subject for this specific invocation
              // so you are not affecting the Subject set on the thread. You may
              // keep any custom objects that you want to propagate in the Subject.
              // This example removes everything and adds just the new information
              // back in.
     try
     {
      subject.getPublicCredentials().clear();
      subject.getPrivateCredentials().clear();
      subject.getPrincipals().clear();
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

              // Adds everything from basic_auth_subject into the login subject.
              // This completes the mapping to the new user.
     try
     {
      subject.getPublicCredentials().addAll(basic_auth_subject.
                  getPublicCredentials());
      subject.getPrivateCredentials().addAll(basic_auth_subject.
                  getPrivateCredentials());
      subject.getPrincipals().addAll(basic_auth_subject.
                  getPrincipals());
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
```

```
   }
   catch (PrivilegedActionException e)
   {
    throw new WSLoginFailedException (e.getException().getMessage(),
              e.getException());
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.csiv2.CSIv2PerformPolicy csiv2PerformPolicy = null;
 javax.security.auth.Subject basic_auth_subject = null;
}
```

Common Secure Interoperability Version 2 and Security Authentication Service client configuration

A secure Java client requires configuration properties to determine how to perform security with a server.

These configuration properties are typically put into a properties file somewhere on the client system and referenced by specifying the following system property on the command line of the Java client. For example, this property accepts any valid Web address.

```
-Dcom.ibm.CORBA.ConfigURL=file:profile_root/properties/sas.client.props
```

When this file is processed by the Object Request Broker (ORB), security can be enabled between the Java client and the target server.

If any syntax problems exist with the ConfigURL property and the `sas.client.props` file is not found, the Java client proceeds to connect insecurely. Errors display indicating the failure to read the ConfigURL property. Typically the problem is related to having two slashes after `file`, which is not valid.

V6.0.x Use the following properties to configure the SAS and CSIv2 authentication protocols:

-
- "Security Authentication Service authentication protocol client settings" on page 1328

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Common authentication protocol settings for a client configuration:

You can use settings in the sas.client.props file to configure Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2) clients.

V6.0.x Use the following settings in the *app_server_root*/properties/sas.client.props file to configure SAS and CSIv2 clients.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

com.ibm.CORBA.securityEnabled

Use to determine if security is enabled for the client process.

Setting	Value
Data Type	Boolean
Default	True

Setting	Value
Valid values	True or false

com.ibm.CSI.protocol

Use to determine which authentication protocols are active.

The client can configure protocols of ibm, csiv2 or both as active. The only possible values for an authentication protocol are ibm, csiv2 and both. Do not use sas for the value of an authentication protocol. This restriction applies to both client and server configurations. The following list provides information about using each of these protocol options:

ibm Use this authentication protocol option when you are communicating with WebSphere Application Server Version 4.x or earlier servers.

V6.0.x **csiv2**
Use this authentication protocol option when you are communicating with WebSphere Application Server Version 5 or later servers because the SAS interceptors are not loaded and running for each method request.

both Use this authentication protocol option for interoperability between WebSphere Application Server Version 4.x or earlier servers and WebSphere Application Server Version 5 or later servers. Typically, specifying both provides greater interoperability with other servers.

Setting	Value
Data type	String
Default	Both
Valid values	ibm, csiv2, both

com.ibm.CORBA.authenticationTarget

Use to determine the type of authentication mechanism for sending security information from the client to the server.

If basic authentication is specified, the user ID and password are sent to the server. Using the Secure Sockets Layer (SSL) transport with this type of authentication is recommended; otherwise, the password is not encrypted. The target server must support the specified authentication target.

If you specify Lightweight Third Party Authentication (LTPA), then LTPA must be the mechanism configured at the server for a method request to proceed securely.

Setting	Value
Data type	String
Default	BasicAuth
Valid values	BasicAuth, LTPA

com.ibm.CORBA.validateBasicAuth

Use to determine if the user ID and password get validated immediately after the login data is entered when the authenticationTarget property is set to BasicAuth.

In previous releases, BasicAuth logins validated only with the initial method request. During the first request, the user ID and password are sent to the server. This request is the first time that the client can notice an error, if the user ID or password is incorrect. The validateBasicAuth method is specified and the validation of the user ID and password occurs immediately to the security server.

For performance reasons, you might want to disable this property if you do not want to verify the user ID and password immediately. If the client program can wait, it is better to have the initial method request flow to the user ID and password. However, program logic might not be this simple because of error handling considerations.

Setting	Value
Data type	Boolean
Default	True
Valid values	True, False

com.ibm.CORBA.authenticationRetryEnabled

Use to specify that a failed login attempt is retried. This property determines if a retry occurs for other errors, such as stateful sessions that are not found on a server or validation failures at the server because of an expiring credential.

The minor code in the exception that is returned to a client determines which errors are retried. The number of retry attempts is dependent upon the com.ibm.CORBA.authenticationRetryCount property.

Setting	Value
Data type	Boolean
Default	True
Valid values	True, False

com.ibm.CORBA.authenticationRetryCount

Use to specify the number of retries that occur until either a successful authentication occurs or the maximum retry value is reached.

When the maximum retry value is reached, the authentication exception is returned to the client.

Setting	Value
Data type	Integer
Default	3
Range	1-10

com.ibm.CORBA.loginSource

Use to specify how the request interceptor attempts to log in if it does not find an invocation credential already set.

This property is valid only if message layer authentication occurs. If only transport layer authentication occurs, this property is ignored. When specifying properties, the following two additional properties must be defined:

- com.ibm.CORBA.loginUserid

- com.ibm.CORBA.loginPassword

When performing a programmatic login, it is not necessary to specify none as the login source. The request fails if a credential is set as the invocation credential during a method request.

Setting	Value
Data type	String
Default	Prompt
Valid values	Prompt, key file, stdin, none, properties

com.ibm.CORBA.loginUserid

Use to specify the user ID when a properties login is configured and message layer authentication occurs.

This property is valid only when com.ibm.CORBA.loginSource=properties. Also set the com.ibm.CORBA.loginPassword property.

Setting	Value
Data type	String
Range	Any string that is appropriate for a user ID in the configured user registry of the server.

com.ibm.CORBA.loginPassword

Use to specify the password when a properties login is configured and message layer authentication occurs.

This property is valid only when com.ibm.CORBA.loginSource=properties. Also set the com.ibm.CORBA.loginUserid property.

Setting	Value
Data type	String
Range	Any string that is appropriate for a password in the configured user registry of the server.

com.ibm.CORBA.keyFileName

Use to specify the key file that is used to log in.

A key file is a file that contains a list of realm, user ID, and password combinations that a client uses to log into multiple realms. The realm that is used is the one found in the interoperable object reference (IOR) for the current method request. The value of this property is used when the com.ibm.CORBA.loginSource=key file is used.

Setting	Value
Data type	String
Default	C;/WebSphere/AppServer/properties/wsserver.key
Range	Any fully qualified path and file name of a WebSphere Application Server key file.

com.ibm.CORBA.loginTimeout

Use to specify the length of time that the login prompt stays available before it is considered a failed login.

Setting	Value
Data type	Integer
Units	Seconds
Default	300 (5 minute intervals)
Range	0 - 600 (10 minute intervals)

com.ibm.CORBA.securityEnabled

Use to determine if security is enabled for the client process.

Setting	Value
Data type	Boolean
Default	True
Range	True, False

Related tasks

"Configuring RMI over IIOP" on page 1302
Complete the following steps to configure Common Secure Interoperability Version 2 (CSIV2) and Security Authentication Service (SAS).

Related reference

"Common Secure Interoperability Version 2 and Security Authentication Service client configuration" on page 1324
A secure Java client requires configuration properties to determine how to perform security with a server.

Security Authentication Service authentication protocol client settings:

In addition to those properties which are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2), this article documents properties which are valid only for the SAS authentication protocol.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

com.ibm.CORBA.standardPerformQOPModels

Specifies the strength of the ciphers when making an Secure Sockets Layer (SSL) connection.

Data type: String
Default: High
Range Low, Medium, High

Java Authentication and Authorization Service

The standard Java 2 security application programming interface (API) helps enforce access control based on the location of the code source or the author or packager of the code that signed the jar file. The current principal of the running thread is not considered in the Java 2 security authorization. Instances where authorization is based on the principal, as opposed to the code base, and the user exist. The Java

Authentication and Authorization Service is a standard Java API that supports the Java 2 security authorization to extend the code base on the principal as well as the code base and users.

The Java Authentication and Authorization Service (JAAS) Version 1.0 extends the Java 2 security architecture of the Java 2 platform with additional support to authenticate and enforce access control with principals and users. JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and extends the access control architecture of the Java 2 platform in a compatible fashion to support user-based authorization or principal-based authorization. WebSphere Application Server fully supports the JAAS architecture. JAAS extends the access control architecture to support role-based authorization for Java 2 Platform, Enterprise Edition (J2EE) resources including servlets, JavaServer Pages (JSP) files, and Enterprise JavaBeans (EJB) components.

Refer to “Java 2 security” on page 1061 for more information.

The following sections cover the JAAS implementation and programming model:
- Login configuration for Java Authentication and Authorization Service
- “Programmatic login” on page 1357
- “Java Authentication and Authorization Service authorization”

The JAAS documentation can be found at http://www.ibm.com/developerworks/java/jdk/security. Scroll down to find the JAAS documentation for your platform.

Java Authentication and Authorization Service authorization

Java 2 security architecture uses a security policy to specify which access rights are granted to running code. This architecture is *code-centric*. The permissions are granted based on code characteristics including where the code is coming from, whether it is digitally signed, and by whom. Authorization of the Java Authentication and Authorization Service (JAAS) augments the existing code-centric access controls with new user-centric access controls. Permissions are granted based on what code is running and who is running it.

When using JAAS authentication to authenticate a user, a *subject* is created to represent the authenticated user. A subject is comprised of a set of principals, where each principal represents an identity for that user. You can grant permissions in the policy to specific principals. After the user is authenticated, the application can associate the subject with the current access control context. For each subsequent security-checked operation, the Java runtime automatically determines whether the policy grants the required permission to a specific principal only. If so, the operation is supported if the subject that is associated with the access control context contains the designated principal only.

Associate a subject with the current access control context by calling the static doAs method from the subject class, passing it an authenticated subject and the java.security.PrivilegedAction or java.security.PrivilegedExceptionAction method. The doAs method associates the provided subject with the current access control context and then invokes the run method from the action. The run method implementation contains all the code that ran as the specified subject. The action runs as the specified subject.

In the Java 2 Platform, Enterprise Edition (J2EE) programming model, when invoking the Enterprise JavaBeans (EJB) method from an enterprise bean or servlet, the method runs under the user identity that is determined by the run-as setting. The J2EE Version 1.4 Specification does not indicate which user identity to use when invoking an enterprise bean from a Subject.doAs action block within either the EJB code or the servlet code. A logical extension is to use the proper identity that is specified in the subject when invoking the EJB method within the Subject doAs action block.

Letting the Subject.doAs action overwrite the run-as identity setting is an ideal way to integrate the JAAS programming model with the J2EE run-time environment. However, JAAS introduced an issue into the Software Development Kit (SDK), Java Technology Edition Versions 1.3 or later when integrating the JAAS Version 1.0 or later implementation with the Java 2 security architecture. A subject, which is associated with the access control context is cut off by a doPrivileged call when a doPrivileged call occurs within the

Subject.doAs action block. Until this problem is corrected, no reliable and run-time efficient way is available to guarantee the correct behavior of Subject.doAs action in a J2EE run-time environment.

The problem can be explained better with the following example:

```
Subject.doAs(subject, new java.security.PrivilegedAction() {
   Public Object run() {
       // Subject is associated with the current thread context
       java.security.AccessController.doPrivileged( new
           java.security.PrivilegedAction() {
                       public Object run() {
                       // Subject was cut off from the current
                       // thread context
   return null;
               }
        });
        // Subject is associated with the current thread context
        return null;
   }
});
```

In the previous code example, the Subject object is associated with the context of the current thread. Within the run method of a doPrivileged action block, the Subject object is removed from the thread context. After leaving the doPrivileged block, the Subject object is restored to the current thread context. Because doPrivileged blocks can be placed anywhere along the running path and instrumented quite often in a server environment, the run-time behavior of a doAs action block becomes difficult to manage.

To resolve this difficulty, WebSphere Application Server provides a WSSubject helper class to extend the JAAS authorization to a J2EE EJB method invocation, as described previously. The WSSubject class provides static doAs and doAsPrivileged methods that have identical signatures to the subject class. The WSSubject.doAs method associates the Subject to the currently running thread. The WSSubject.doAs and WSSubject.doAsPrivileged methods then invoke the corresponding Subject.doAs and Subject.doAsPrivileged methods. The original credential is restored and associated with the running thread upon leaving the WSSubject.doAs and WSSubject.doAsPrivileged methods.

The WSSubject class is not a replacement of the subject object, but rather a helper class to ensure consistent run-time behavior as long as an EJB method invocation is a concern.

The following example illustrates the run-time behavior of the WSSubject.doAs method:

```
WSSubject.doAs(subject, new java.security.PrivilegedAction() {
   Public Object run() {
       // Subject is associated with the current thread context
       java.security.AccessController.doPrivileged( new
           java.security.PrivilegedAction() {
                      public Object run() {
                         // Subject was cut off from the current thread
                         // context.
   return null;
               }
        });
        // Subject is associated with the current thread context
   return null;
   }
});
```

The Subject.doAs and Subject.doAsPrivileged methods are not integrated with the J2EE run-time environment. EJB methods that are invoked within the Subject.doAs and Subject.doAsPrivileged action blocks run under the identity that is specified by the run-as setting and not by the subject identity.

- The Subject object that is generated by the WSLoginModuleImpl instance and the WSClientLoginModuleImpl instance contains a principal that implements the WSPrincipal interface. Using the getCredential method for a WSPrincipal object returns an object that implements the WSCredential interface. You can also find the WSCredential object instance in the PublicCredentials list of the subject instance. Retrieve the WSCredential object from the PublicCredentials list instead of using the getCredential method.
- The getCallerPrincipal method for the WSSubject class returns a string that represents the caller security identity. The return type differs from the getCallerPrincipal method of the java.security.Principal EJBContext interface.
- The Subject object that is generated by the Java 2 Connector (J2C) DefaultPrincipalMapping module contains a resource principal and a PasswordCredentials list. The resource principal represents the RunAs identity.

For more information, see “J2EE connector security” on page 682.

Using the Java Authentication and Authorization Service programming model for Web authentication

WebSphere Application Server supports the Java 2 Platform, Enterprise Edition (J2EE) declarative security model. You can define the authentication and access control policy using the J2EE deployment descriptor. You can further stack custom login modules to customize the WebSphere Application Server authentication mechanism.

A custom login module can perform principal and credential mapping, custom security token and custom credential-processing, and error-handling among other possibilities. Typically, you do not need to use application code to perform authentication function. Use the programming techniques that are described in this section if you have to perform authentication function in application code. For example, if you have applications that programmed to the SSOAuthenticator helper function, you can use the following programming interface. The SSOAuthenticator helper function was deprecated starting with WebSphere Application Server Version 4.0. Use declarative security as a rule; use the techniques that are described in this section as a last resort.

When the Lightweight Third-Party Authentication (LTPA) mechanism single sign-on (SSO) option is enabled, the Web client login session is tracked by an LTPA SSO token cookie after successful login. At logout, this token is deleted to terminate the login session, but the server-side subject is not deleted. When you use the declarative security model, the WebSphere Application Server Web container performs client authentication and login session management automatically. You can perform authentication in application code by setting a login page without a J2EE security constraint and by directing client requests to your login page first. Your login page can use the Java Authentication and Authorization Service (JAAS) programming model to perform authentication. To enable WebSphere Application Server Web login modules to generate SSO cookies, use the following steps.

1. Select the `wsMapDefaultInboundLoginModule` login module and click **Custom properties**. There are two login modules defined in your login configuration: `ltpaLoginModule` and `wsMapDefaultInboundLoginModule`.
2. Create a new system login JAAS configuration. To access the panel, click **Security > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **System logins**.
3. Manually clone the WEB_INBOUND login configuration, and give it a new alias. To clone the login configuration, click **New**, enter a name for the configuration, click **Apply**, then click **JAAS login modules** under Additional properties. Click **New** and configure the JAAS login module. For more information, see Login module settings for Java Authentication and Authorization Service. WebSphere Application Server Web container uses the WEB_INBOUND login configuration to authenticate Web

clients. Changing the WEB_INBOUND login configuration affects all Web applications in the cell. You should create your own login configuration by cloning the contents of the WEB_INBOUND login configuration.

4. Select the `wsMapDefaultInboundLoginModule` login module and click **Custom properties**. There are two login modules defined in your login configuration: `ltpaLoginModule` and `wsMapDefaultInboundLoginModule`.
5. Add a login property name *cookie* with a value of **true**. The two login modules are enabled to generate LTPA SSO cookies. Do not add the cookie login option to the original WEB_INBOUND login configuration.
6. Stack your custom LoginModule(s) in the new login configuration (optional).
7. Use your login page for programmatic login by perform a JAAS LoginContext.login using your newly defined login configuration. After a successful login, either the `ltpaLoginModule` or the `wsMapDefaultInboundLoginModule` generates an LTPA SSO cookie upon a successful authentication. Exactly which LoginModule generates the SSO cookie depends on many factors, including system authentication configuration and runtime condition (which is beyond the scope of this section).
8. Call the modified `WSSubject.setRunAsSubject` method to add the subject to the authentication cache. The subject must be a WebSphere Application Server JAAS subject created by `LoginModule`. Adding the subject to the authentication cache recreates a subject from SSO token.
9. Use your programmatic logout page to revoke SSO cookies by invoking the `revokeSSOCookies` method from the WSSecurityHelper class. The term cookies is used because WebSphere Application Server Release 5.1.1 (and later) release supports a new LTPA SSO token with a different encryption algorithm, but can be configured to generate the original LTPA SSO token for backward compatibility. Note that the subject is still in the authentication cache and only the SSO cookies are revoked.

Use the following code sample to perform authentication:

```
Suppose you wrote a LoginServlet.java:

 Import com.ibm.wsspi.security.auth.callback.WSCallbackHandlerFactory;
 Import com.ibm.websphere.security.auth.WSSubject;

 public Object login(HttpServletRequest req, HttpServletResponse res)
 throws ServletException {

 PrintWriter out = null;
 try {
  out =  res.getWriter();
       res.setContentType("text/html");
 } catch (java.io.IOException e){
  // Error handling
 }

 Subject subject = null;
 try {
 LoginContext lc = new LoginContext("system.Your_login_configuration",
WSCallbackHandlerFactory.getInstance().getCallbackHandler(
userid, null, password, req, res, null));
  lc.login();
  subject = lc.getSubject();
       WSSubject.setRunAsSubject(subject);
 } catch(Exception e) {
  // catch all possible exceptions if you want or handle them separately
  out.println("Exception in LoginContext login + Exception = " +
 e.getMessage());
  throw new ServletException(e.getMessage());
 }

The following is sample code to revoke the SSO cookies upon a programming logout:


The LogoutServlet.java:
```

```
public void logout(HttpServletRequest req, HttpServletResponse res,
Object retCreds)  throws ServletException {
  PrintWriter out =null;
  try {
   out =  res.getWriter();
       res.setContentType("text/html");
  } catch (java.io.IOException e){
  // Error Handling
  }
  try {
    WSSecurityHelper.revokeSSOCookies(req, res);
  } catch(Exception e) {
   // catch all possible exceptions if you want or handle them separately
   out.println("JAASLogoutServlet: logout Exception = " + e.getMessage());
   throw new ServletException(e);
  }
 }
```

For more information on JAAS authentication, refer to Developing programmatic logins with the Java Authentication and Authorization Service. For more information on the AuthenLoginModule login module, refer to Example: Customizing a server-side Java Authentication and Authorization Service authentication and login configuration.

Authorizing access to resources

WebSphere Application Server provides many different methods for authorizing accessing resources. For example, you can assign roles to users and configure a built-in or external authorization provider.

You can create an application, an Enterprise JavaBeans (EJB) module, or a Web module and secure them using assembly tools.

To authorize user or group access to resources, read the following articles:

1. Secure you application during assembly and deployment. For more information on how to create a secure application using an assembly tool, such as the IBM Rational Application Developer, see Securing applications during assembly and deployment.

   For general information about the tools that WebSphere Application Server supports, see Assembly tools and Assembling applications.
2. Authorize access to Java 2 Platform, Enterprise Edition (J2EE) resources. WebSphere Application Server supports authorization that is based on the Java Authorization Contract for Containers (JACC) specification in addition to the default authorization. When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified. For more information, see “Authorization providers” on page 1343.
3. Authorize access to administrative resources. You can assign users and groups to predefined administrative roles such as the monitor, configurator, operator, administrator, and iscadmins roles. These roles determine which tasks a user can perform in the administrative console. For more information, see “Authorizing access to administrative roles” on page 1386.

After authorizing access to resources, configure the Application Server for secure communication. For more information, see “Securing communications” on page 1394.

Authorization technology

Authorization information determines whether a user or group has the necessary privileges to access resources.

WebSphere Application Server supports many authorization technologies including the following:

- Authorization involving the Web container and Java 2 Platform, Enterprise Edition (J2EE) technology

- Authorization involving an enterprise bean application and J2EE technology
- Authorization involving Web services and J2EE technology
- Java Message Service (JMS)
- Java Authorization Contract for Containers (JACC)

  WebSphere Application Server supports both a default authorization provider, which was supported in previous releases, and an authorization provider that is based on the Java Authorization Contract for Containers (JACC) specification. The JACC-based authorization provider enables third-party security providers to handle the J2EE authorization. For more information, see “JACC support in WebSphere Application Server” on page 1344.
- Java Authentication and Authorization Service (JAAS)

  For more information, see “Java Authentication and Authorization Service” on page 1328.
- Java 2 security

  For more information, see “Java 2 security” on page 1061.
- Naming and administrative authorization
- Pluggable authorization

  WebSphere Application Server supports an authorization infrastructure that enables you to plug in an external authorization provider. For more information, see “Enabling an external JACC provider” on page 1368.

Administrative roles and naming service authorization

WebSphere Application Server extends the Java 2 Platform, Enterprise Edition (J2EE) security role-based access control to protect the product administrative and naming subsystems.

Administrative roles

A number of administrative roles are defined to provide the degrees of authority that are needed to perform certain WebSphere Application Server administrative functions from either the administrative console or the system management scripting interface called wsadmin. The authorization policy is only enforced when administrative security is enabled. The following table describes the administrative roles:

Table 19. Administrative roles that are available through the administrative console and wsadmin

Role	Description
Monitor	An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks: • View the WebSphere Application Server configuration. • View the current state of the Application Server.
Configurator	An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the day-to-day configuration tasks. For example, a configurator can complete the following tasks: • Create a resource. • Map an application server. • Install and uninstall an application. • Deploy an application. • Assign users and groups-to-role mapping for applications. • Set up Java 2 security permissions for applications. • Customize the Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations. **Important:** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Table 19. Administrative roles that are available through the administrative console and wsadmin (continued)

Role	Description
Operator	An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks: • Stop and start the server. • Monitor the server status in the administrative console.
Administrator	An individual or group that uses the administrator role has the operator and configurator privileges plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks: • Modify the server user ID and password. • Configure authentication and authorization mechanisms. • Enable or disable administrative security. **Note:** In previous releases of WebSphere Application Server, the **Enable administrative security** option is known as the **Enable global security** option. • Enforce Java 2 security using the **Use Java 2 security to restrict application access to local resources** option. • Change the Lightweight Third Party Authentication (LTPA) password and generate keys. • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration. **Note:** An administrator cannot map users and groups to the administrator roles.
AdminSecurityManager	Only users who are granted this role can map users to administrative roles. Also, when fine-grained administrative security is used, only users who are granted this role can manage authorization groups. See “Administrative roles” on page 1340 for more information.

Table 20. Additional administrative role that is available through the administrative console

Role	Description
iscadmins	This role is only available for administrative console users and not for wsadmin users. Users who are granted this role have administrator privileges for managing users and groups in the federated respositories. For example, a user of the iscadmins role can complete the following tasks: • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration.

Table 21. Additional administrative role that is available through wsadmin

Role	Description
Deployer	This role is only available for wsadmin users and not for administrative console users. Users who are granted this role can perform both configuration actions and run-time operations on applications.

When administrative security is enabled, the administrative subsystem role-based access control is enforced. The administrative subsystem includes the security server, the administrative console, the wsadmin scripting tool, and all the Java Management Extensions (JMX) MBeans. When administrative security is enabled, both the administrative console and the administrative scripting tool require users to provide the required authentication data. Moreover, the administrative console is designed so the control functions that display on the pages are adjusted, according to the security roles that a user has. For example, a user who has only the monitor role can see only the non-sensitive configuration data. A user with the operator role can change the system state.

When you are changing registries (for example, from a federated repository to LDAP), make sure you remove the information that pertains to the previously configured registry for console users and console groups.

When administrative security is enabled, WebSphere Application Servers run under the server identity that is defined under the active user registry configuration. Although it is not shown on the administrative console and in other tools, a special Server subject is mapped to the administrator role. The WebSphere Application Server runtime code, which runs under the server identity, requires authorization to runtime operations. If no other user is assigned administrative roles, you can log into the administrative console or to the wsadmin scripting tool using the server identity to perform administrative operations and to assign other users or groups to administrative roles. Because the server identity is assigned to the administrative role by default, the administrative security policy requires the administrative role to perform the following operations:

- Change server ID and server password
- Enable or disable WebSphere Application Serveradministrative security
- Enforce Java 2 security using the **Use Java 2 security to restrict application access to local resources** option.
- Change the LTPA password or generate keys
- Assign users and groups to administrative roles

Primary administrative user name

The Version 6.1 release of WebSphere Application Server requires an administrative user, distinguished from the server user identity, to improve auditability of administrative actions. The user name specifies a user with administrative privileges that is defined in the local operating system.

Server user identity

The Version 6.1 release of WebSphere Application Server distinguishes the server identity from the administrative user identity to improve auditability. The server user identity is used for authenticating server-to-server communications.

Internal server ID

The internal server ID enables the automatic generation of the user identity for server-to-server authentication. Automatic generation of the server identity supports improved auditability for cells only for Version 6.1 or later nodes. In the Version 6.1 release of WebSphere Application Server, you can save the internally-generated server ID because the Security WebSphere Common Configuration Model (WCCM) model contains a new tag, internalServerId. You do not need to specify a server user ID and a password during security configuration except in a mixed-cell environment. An internally-generated server ID adds a further level of protection to the server environment because the server password is not exposed as it is in releases prior to Version 6.1. However, to maintain backwards compatibility, you must specify the server user ID if you use earlier versions of WebSphere Application Server.

When enabling security, you can assign one or more users and groups to naming roles. For more information, see Assigning users to naming roles. However, before assigning users to naming roles, configure the active user registry. User and group validation depends on the active user registry. For more information, see Configuring user registries.

Special subject

In addition to mapping users or groups, you can map a *special-subject* to the administrative roles. A special-subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the administrative role ensures that the user making the request is at least authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action as if security is not enabled.

Naming service authorization

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. These functions affect the content of the WebSphere Application Server name space. Generally, you have two ways in which client programs result in CosNaming calls. The first is through the Java Naming and Directory Interface (JNDI) call. The second is with common object request broker architecture (CORBA) clients invoking CosNaming methods directly.

Four security roles are introduced :

- CosNamingRead
- CosNamingWrite
- CosNamingCreate
- CosNamingDelete

The roles have authority levels from low to high:

CosNamingRead
: You can query the WebSphere Application Server name space, using, for example, the JNDI lookup method. The special-subject, Everyone, is the default policy for this role.

CosNamingWrite
: You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. As a default policy, Subjects are not assigned this role.

CosNamingCreate
: You can create new objects in the name space through such operations as JNDI createSubcontext and CosNamingWrite operations. As a default policy, Subjects are not assigned this role.

CosNamingDelete
: You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. As a default policy, Subjects are not assigned this role.

Additionally, a Server special-subject is assigned to all of the four CosNaming roles by default. The Server special-subject provides a WebSphere Application Server process, which runs under the server identity, to access all the CosNaming operations. The Server special-subject does not display and cannot be modified through the administrative console or other administrative tools.

Special configuration is not required to enable the server identity as specified when enabling administrative security for administrative use because the server identity is automatically mapped to the administrator role.

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, a server restart is required for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

The CosNaming authorization policy is only enforced when administrative security is enabled. When administrative security is enabled, attempts to do CosNaming operations without the proper role assignment result in an org.omg.CORBA.NO_PERMISSION exception from the CosNaming server.

Each CosNaming function is assigned to only one role. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they have also been assigned CosNamingRead. And in most cases a creator needs to be assigned three roles: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for

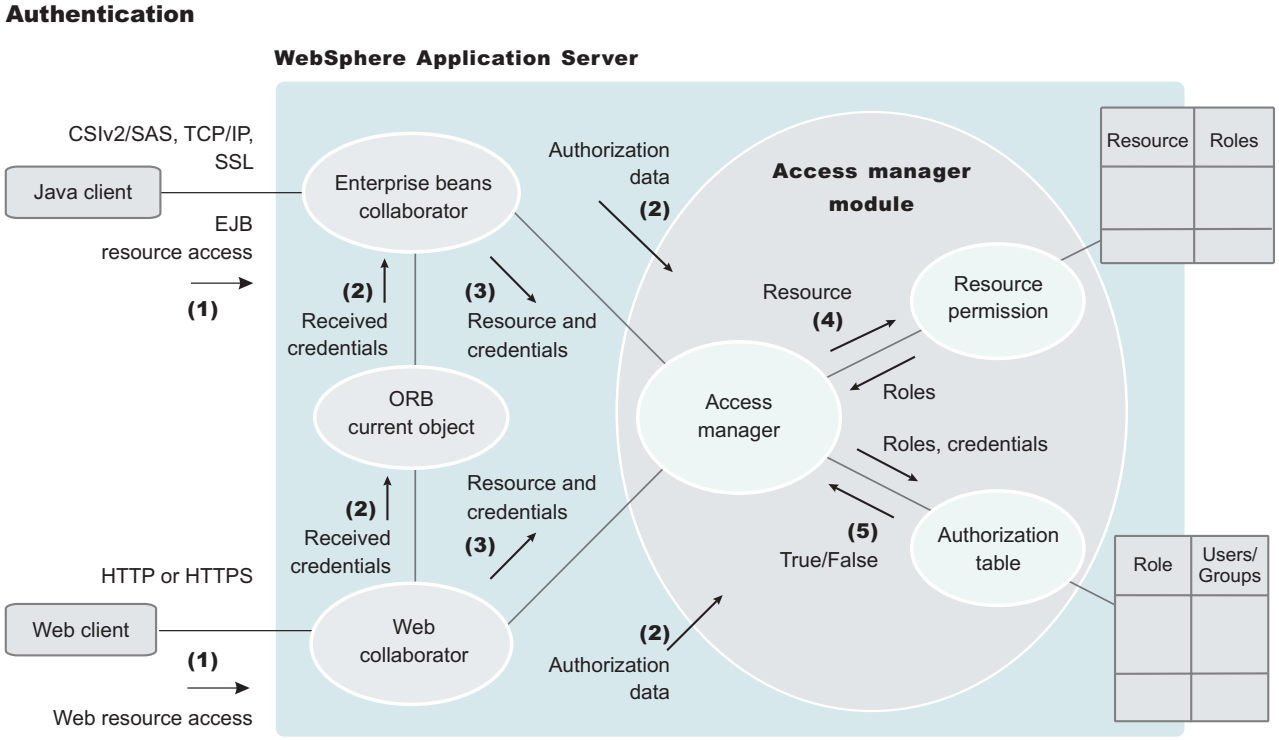
the creator example are included in the CosNamingCreate role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, unexpected org.omg.CORBA.NO_PERMISSION exceptions can occur at runtime. Typically, J2EE applications access the name space and the identity they use is that of the user that authenticated to WebSphere Application Server when accessing the J2EE application. Unless the J2EE application provider clearly communicates the expected naming roles, use caution when changing the default naming authorization policy.

Role-based authorization

Use authorization information to determine whether a caller has the necessary privileges to request a service.

The following figure illustrates the process that is used during authorization.



Web resource access from a Web client is handled by a Web collaborator. The Enterprise JavaBeans (EJB) resource access from a Java client, whether an enterprise bean or a servlet, is handled by an EJB collaborator. The EJB collaborator and the Web collaborator extract the client credentials from the object request broker (ORB) current object. The client credentials are set during the authentication process as received credentials in the ORB current object. The resource and the received credentials are presented to the WSAccessManager access manager to check whether access is permitted to the client for accessing the requested resource.

The access manager module contains two main modules:

- The resource permission module helps determine the required roles for a given resource. This module uses a resource-to-roles mapping table that is built by the security runtime during application startup. To build the resource-to-role mapping table, the security runtime reads the deployment descriptor of the enterprise beans or the Web module (`ejb-jar.xml` file or `web.xml` file)

- The authorization table module consults a role-to-user or group table to determine whether a client is granted one of the required roles. The role-to-user or group mapping table, also known as the *authorization table*, is created by the security runtime during application startup.

  To build the authorization table, the security runtime reads the `ibm-application-bnd.xmi` application binding file.

Use authorization information to determine whether a caller has the necessary privilege to request a service. You can store authorization information many ways. For example, with each resource, you can store an access-control list, which contains a list of users and user privileges. Another way to store the information is to associate a list of resources and the corresponding privileges with each user. This list is called a *capability list*.

WebSphere Application Server uses the Java 2 Platform, Enterprise Edition (J2EE) authorization model. In this model, authorization information is organized as follows:

During the assembly of an application, permission to invoke methods is granted to one or more roles. A role is a set of permissions; for example, in a banking application, roles can include teller, supervisor, clerk, and other industry-related positions. The teller role is associated with permissions to run methods that are related to managing the money in an account, such as the withdraw and deposit methods. The teller role is not granted permission to close accounts; this permission is given to the supervisor role. The application assembler defines a list of method permissions for each role. This list is stored in the deployment descriptor for the application.

Two *special subjects* are not defined by the J2EE model: AllAuthenticatedUsers and Everyone. A special subject is a product-defined entity that is independent of the user registry. This entity is used to generically represent a class of users or groups in the registry.
- The AllAuthenticatedUsers subject permits all authenticated users to access protected methods. As long as the user can authenticate successfully, the user is permitted access to the protected resource. All of this is done independent of the user registry.
- Everyone is a special subject that permits unrestricted access to a protected resource. Users do not have to authenticate to get access; this special subject provides access to protected methods as if the resources are unprotected. All of this is done independent of the user registry.

During the deployment of an application, real users or groups of users are assigned to the roles. When a user is assigned to a role, the user gets all the method permissions that are granted to that role.

The application deployer does not need to understand the individual methods. By assigning roles to methods, the application assembler simplifies the job of the application deployer. Instead of working with a set of methods, the deployer works with the roles, which represent semantic groupings of the methods.

Users can be assigned to more than one role; the permissions that are granted to the user are the union of the permissions granted to each role. Additionally, if the authentication mechanism supports the grouping of users, these groups can be assigned to roles. Assigning a group to a role has the same effect as assigning each individual user to the role.

A best practice during deployment is to assign groups instead of individual users to roles for the following reasons:
- Improves performance during the authorization check. Typically far fewer groups exist than users.
- Provides greater flexibility, by using group membership to control resource access.
- Supports the addition and deletion of users from groups outside of the product environment. This action is preferred to adding and removing them to WebSphere Application Server roles. Stop and restart the enterprise application for these changes to take effect. This action can be very disruptive in a production environment.

At runtime, WebSphere Application Server authorizes incoming requests based on the user's identification information and the mapping of the user to roles. If the user belongs to any role that has permission to run a method, the request is authorized. If the user does not belong to any role that has permission, the request is denied.

The J2EE approach represents a declarative approach to authorization, but it also recognizes that you cannot deal with all situations declaratively. For these situations, methods are provided for determining user and role information programmatically. For enterprise beans, the following two methods are supported by WebSphere Application Server:
- **getCallerPrincipal**: This method retrieves the user identification information.
- **isCallerInRole**: This method checks the user identification information against a specific role.

For servlets, the following methods are supported by WebSphere Application Server:
- getRemoteUser
- isUserInRole
- getUserPrincipal

These methods correspond in purpose to the enterprise bean methods.

For more information on the J2EE security authorization model, see the following Web site: http://java.sun.com

Administrative roles

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

A number of administrative roles are defined to provide degrees of authority that are needed to perform certain administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when administrative security is enabled. The following table describes the administrative roles:

Administrative roles

Role	Description
Monitor	An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks: • View the WebSphere Application Server configuration. • View the current state of the Application Server.
Configurator	An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the day-to-day configuration tasks. For example, a configurator can complete the following tasks: • Create a resource. • Map an application server. • Install and uninstall an application. • Deploy an application. • Assign users and groups-to-role mapping for applications. • Set up Java 2 security permissions for applications. • Customize the Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations. **Important:** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Administrative roles

Operator	An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks: • Stop and start the server. • Monitor the server status in the administrative console.
Administrator	An individual or group that uses the administrator role has the operator and configurator privileges plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks: • Modify the server user ID and password. • Configure authentication and authorization mechanisms. • Enable or disable administrative security. • Enable or disable Java 2 security. • Change the Lightweight Third Party Authentication (LTPA) password and generate keys. • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration. **Note:** An administrator cannot map users and groups to the administrator roles.
iscadmins	This role is only available for administrative console users, not for wsadmin users. Users who are granted this role have administrator privileges for managing users and groups in the federated repositories. For example, a user of the iscadmins role can complete the following tasks: • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration.
Deployer	This role is only available for wsadmin users, not for administrative console users. Users granted this role can perform both configuration actions and runtime operations on applications. See the “Deployer role” section for more details.
AdminSecurityManager	This role is only available for wsadmin users, not for administrative console users. When using wsadmin, users granted this role can map users to administrative roles. Also, when fine grained admin security is used, users granted this role can manage authorization groups. See the “AdminSecurityManager role” on page 1342 section for more details.

The server ID that is specified and the administrative ID, if specified, when enabling administrative security is automatically mapped to the administrator role.

Users and groups can be added or removed from the administrative roles from the WebSphere Application Server administrative console at any time. A best practice is to map a group or groups, rather than specific users, to administrative roles because it is more flexible and easier to administer.

In addition to mapping user or groups, a special-subject can also be mapped to the administrative roles. A special-subject subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the administrative role ensures that the user making the request is at least authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if security was not enabled.

Deployer role

A user that is granted a deployer role can perform all of the configuration and runtime operations on an application. A deployer role can be subsets of both configurator and operator roles. However, a user granted a deployer role cannot configure or operate any other resources (server, node).

When fine-grained administrative security is used, only a user granted a deployer role to an application can configure and operate that application.

Cell level configurators can configure applications (install, edit, deploy, and uninstall). Cell level operators can also operate (start and stop) applications. However, a user granted a deployer role at cell level can also perform configuration and operation on all applications.

The following table lists the capabilities of the deployer role when fine-grained administrative security is used:

Operation	Required Roles (Any one)
Install application	Cell-configurator, target-deployer
Uninstall application	Cell-configurator, application-deployer
List application	Cell-monitor, application-monitor
Edit, update and redeploy application	Cell-configurator, application-deployer
Export application	Cell-monitor, application-monitor
Start-stop application	Cell-operator, application-deployer

Where:

Cell-configurator
is the configurator role at cell level.

Application-deployer
is the deployer role for the application that is being managed.

Target-deployer
is the deployer role for all servers or clusters for which an application is targeted. If you have a target-deployer role, you can install a new application on the target. However, to edit or update the installed application, you must be included in the authorization group of the installed application-deployer.

The target-deployer can not explicitly start or stop a new application. However, when a target-deployer starts a server on a target, all of the applications that have their auto-start attribute set to `yes` are started when the server starts.

It is recommended that the application-deployer set this attribute to `true` if the application-deployer does not want the application to be started by the target-deployer.

AdminSecurityManager role

The AdminSecurityManager role separates administrative security administration from other application administration.

By default, serverId and adminID, if specified, are assigned to this role in the cell level authorization table. This role implies a monitor role. However, an administrator role does not imply the AdminSecurityManager role.

When fine-grained admin security is used, only a user granted this role at cell level can manage administrative authorization groups. However, a user granted this role for each administrative authorization group can map users to administrative roles for those groups. The following lists the capabilities of the AdminSecurityManager role at different levels (cell and administrative authorization group):

Action	Who can perform
Map users to administrative roles for cell level	Only the AdminSecurityManager of the cell

Action	Who can perform
Map users to administrative roles for an authorization group	Only the AdminSecurityManager of that authorization group or the AdminSecurityManager of the cell
Manage authorization groups (create, delete, add resource to an authorization group, or remove resource from an authorization group or list)	Only the AdminSecurityManager of the cell

Related tasks

"Assigning users to naming roles" on page 1390
Use this task to assign users to naming roles by using the administrative console.

Enterprise bean component security

An Enterprise JavaBeans (EJB) module consists of one or more beans. You can use development tools such as Rational Application Developer to develop an EJB module. You can also enforce security at the EJB method level.

You can assign a set of EJB methods to a set of one or more roles. When an EJB method is secured by associating a set of roles, grant at least one role in that set so that you can access that method. To exclude a set of EJB methods from access mark the set **excluded**. You can give everyone access to a set of enterprise beans methods by clearing those methods. You can run enterprise beans as a different identity, using the runAs identity, before invoking other enterprise beans.

Authorization providers

WebSphere Application Server supports authorization that is based on the Java Authorization Contract for Containers (JACC) specification in addition to the default authorization.

JACC is a new specification in Java 2 Platform, Enterprise Edition (J2EE) 1.4. It enables third-party security providers to manage authorization in the application server.

When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified. The default authorization does not require special setup, and the default authorization engine makes all of the authorization decisions. However, if a JACC provider is configured and set up for WebSphere Application Server to use, all of the enterprise beans and Web authorization decisions are delegated to the JACC provider.

WebSphere Application Server supports security for J2EE applications and also for its administrative components. J2EE applications, such as Web and Enterprise JavaBeans (EJB) components are protected and authorized per the J2EE specification. The administrative components are internal to WebSphere Application Server and are protected by the role-based authorizer. The administrative components include the administrative console, MBeans, and other components such as naming and security. For more information on administrative security, see "Role-based authorization" on page 1338.

When a JACC provider is used for authorization in WebSphere Application Server, all of the J2EE application-based authorization decisions are delegated to the provider per the JACC specification. However, all administrative security authorization decisions are made by the WebSphere Application Server default authorization engine. The JACC provider is not called to make the authorization decisions for administrative security.

When a protected J2EE resource is accessed, the authorization decision to give access to the principal is the same whether using the default authorization engine or a JACC provider. Both of the authorization models satisfy the J2EE specification, and function the same. Choose a JACC provider only when you want to work with an external security provider such as Tivoli Access Manager. In this instance, the security provider must support the JACC specification and be set up to work with WebSphere Application

Server. Setting up and configuring a JACC provider requires additional configuration steps, depending on the provider. Unless you have an external security provider that you can use with WebSphere Application Server, use the default authorization.

JACC support in WebSphere Application Server:

WebSphere Application Server supports the Java Authorization Contract for Containers (JACC) specification, which enables third-party security providers to handle the Java 2 Platform, Enterprise Edition (J2EE) authorization.

The JACC specification requires that both the containers in the application server and the provider satisfy some requirements. Specifically, the containers are required to propagate the security policy information to the provider during the application deployment and to call the provider for all authorization decisions. The providers are required to store the policy information in their repository during application deployment. The providers then use this information to make authorization decisions when called by the container.

JACC access decisions

When security is enabled and an enterprise bean or Web resource is accessed, the Enterprise JavaBeans (EJB) container or Web container calls the security runtime to make an authorization decision on whether to permit access. When using an external provider, the access decision is delegated to that provider.

According to the Java Authorization Contract for Containers (JACC) specification, the appropriate permission object is created, the appropriate policy context handlers are registered, and the appropriate policy context identifier (contextID) is set. A call is made to the java.security.Policy object method that is implemented by the provider to make the access decision.

The following sections describe how the provider is called for both the enterprise bean and the Web resources.

Access decisions for enterprise beans

When security is enabled, and an EJB method is accessed, the EJB container delegates the authorization check to the security runtime. If JACC is enabled, the security runtime uses the following process to perform the authorization check:

1. Creates the EJBMethodPermission object using the bean name, method name, interface name, and the method signature.
2. Creates the context ID and sets it on the thread by using the PolicyContext.setContextID(contextID) method.
3. Registers the required policy context handlers, including the Subject policy context handler.
4. Creates the ProtectionDomain object with principal in the Subject. If no principal exists, null is passed for the principal name.
5. The access decision is delegated to the JACC provider by calling the implies method of the Policy object, which is implemented by the provider. The EJBMethodPermission and the ProtectionDomain objects are passed to this method.
6. The isCallerInRole access check also follows the same process, except that an EJBRoleRefPermission object is created instead of an EJBMethodPermission object.

Access decisions for Web resources

When security is enabled and configured to use a JACC provider, and when a Web resource such as a servlet or a JavaServer Pages (JSP) file is accessed, the security runtime delegates the authorization decision to the JACC provider by using the following process:

1. A WebResourcePermission object is created to see if the URI is cleared. If the provider honors the Everyone subject it is also selected here.
   a. The WebResourcePermission object is constructed with the urlPattern and the HTTP method accessed.
   b. A ProtectionDomain object with a null principal name is created.
   c. The JACC provider Policy.implies method is called with the permission and the protection domain. If the URI access is cleared or given access to the Everyone subject, the provider permits access (return `true`) in the implies method. Access is then granted without further checks.
2. If access is not granted in the previous step, a WebUserDataPermission object is created and used to see if the Uniform Resource Identifier (URI) is precluded, excluded or must be redirected using the HTTPS protocol.
   a. The WebUserDataPermission object is constructed with the urlPattern accessed, the HTTP method invoked, and the transport type of the request. If the request is over HTTPS, the transport type is set to CONFIDENTIAL; otherwise, null is passed.
   b. A ProtectionDomain object with a null principal name is created.
   c. The JACC provider Policy.implies method is called with the permission and the protection domain. If the request is using the HTTPS protocol and the implies method returns `false`, the HTTP 403 error is returned to imply excluded and precluded permission. In this case no further checks are performed. If the request is not using the HTTPS protocol, and the implies returns `false`, the request is redirected over HTTPS.
3. The security runtime attempts to authenticate the user. If the authentication information already exists (for example, LTPA token), it is used. Otherwise, the user's credentials must be entered.
4. After the user credentials are validated, a final authorization check is performed to see if the user is granted access privileges to the URI.
   a. As in Step 1, the WebResourcePermission object is created. The ProtectionDomain object now contains the Principal that is attempting to access the URI. The Subject policy context handler also contains the user's information, which can be used for the access check.
   b. The provider implies method is called using the Permission object and the ProtectionDomain object created previously. If the user is granted permission to access the resource, the implies method returns `true`. If the user is not granted access, the implies method returns `false`.

Even if the order listed previously is changed later (for example, to improve performance) the end result is the same. For example, if the resource is precluded or excluded, the end result is that the resource cannot be accessed.

Using information from the Subject for access decision

If the provider relies on the WebSphere Application Server generated Subject for access decision, the provider can query the public credentials in the Subject to obtain the WSCredential credential. The WSCredential API is used to obtain information about the user, including the name and the groups that the user belongs to. This information is used to make the access decision.

If the provider adds the required information to the Subject, WebSphere Application Server can use the information to make the access decision. The provider might add the information by using the Trust Association Interface feature or by plugging login modules into the Application Server.

The security attribute propagation section contains additional documentation on how to add the WebSphere Application Server required information to the Subject. For more information, see "Propagating security attributes among application servers" on page 1283.

Dynamic module updates in JACC

WebSphere Application Server supports dynamic updates to Web modules under certain conditions. If a Web module is updated, deleted or added to an application, only that module is stopped and started as appropriate. The other existing modules in the application are not impacted, and the application itself is not stopped and then restarted.

When using the default authorization engine, any security policies are modified in the Web modules and the application is stopped and then restarted. When using the Java Authorization Contract for Containers (JACC) based authorization, the behavior depends on the functionality that a provider supports. If a provider can handle dynamic changes to the Web modules, then only the Web modules are impacted. Otherwise, the entire application is stopped and restarted for the new changes in the Web modules to take effect.

A provider can indicate if it supports the dynamic updates by configuring the **Supports dynamic module updates** option in the JACC configuration model (see “Authorizing access to J2EE resources using Tivoli Access Manager” on page 1364 for more information). This option can be enabled or disabled using the administrative console or by scripting. It is expected that most providers store the policy information in their external repository, which makes it possible for them to support these dynamic updates. This option should be enabled by default for most providers.

When the **Supports dynamic module updates** option is enabled, if a Web module that contains security roles is dynamically added, modified, or deleted, only the specific Web modules are impacted and restarted. If the option is disabled, the entire application is restarted. When dynamic updates are performed, the security policy information of the modules impacted are propagated to the provider. For more information about security policy propagation, see “JACC policy propagation” on page 1348.

Initialization of the JACC provider

If a Java Authorization Contract for Containers (JACC) provider requires initialization during server startup, for example, to enable the client code to communicate to the server code, the provider can implement the com.ibm.wsspi.security.authorization.InitializeJACCProvider interface. See “Interfaces that support JACC” on page 1380 for more information.

When this interface is implemented, it is called during server startup. Any custom properties in the JACC configuration model are propagated to the initialize method of this implementation. The custom properties can be entered using either the administrative console or by scripting.

During server shutdown, the cleanup method is called for any clean-up work that a provider requires. Implementation of this interface is strictly optional, and is used only if the provider requires initialization during server startup.

Mixed node environment and JACC

Authorization using Java Authorization Contract for Containers (JACC) is a new feature in WebSphere Application Server Version 6.0.*x*. Also, the JACC configuration is set up at the cell level and is applicable for all the nodes and servers in that cell.

If you are planning to use the JACC-based authorization, the cell must contain Version 6.0.*x* and later nodes only. This restriction implies that a mixed node environment containing a set of Version 5.*x* nodes in a Version 6.0.*x* or later cell is not supported.

JACC providers:

The Java Authorization Contract for Containers (JACC) is a new specification that is introduced in Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 through the Java Specifications Request (JSR) 115 process. This specification defines a contract between J2EE containers and authorization providers.

The contract enables third-party authorization providers to plug into J2EE 1.4 application servers, such as WebSphere Application Server, to make the authorization decisions when a J2EE resource is accessed. The access decisions are made through the standard java.security.Policy object.

In WebSphere Application Server, two authorization contracts are supported using both a native and a third-party JACC provider implementation.

To plug in to WebSphere Application Server, the third-party JACC provider must implement the policy class, policy configuration factory class, and policy configuration interface, which are all required by the JACC specification.

The JACC specification does not specify how to handle the authorization table information between the container and the provider. It is the responsibility of the provider to provide some management facilities to handle this information. The container is not required to provide the authorization table information in the binding file to the provider.

WebSphere Application Server provides the RoleConfigurationFactory and the RoleConfiguration role configuration interfaces to help the provider obtain information from the binding file, as well as an initialization interface (InitializeJACCProvider). The implementation of these interfaces is optional. See “Interfaces that support JACC” on page 1380 for more information about these interfaces.

Tivoli Access Manager as the default JACC provider for WebSphere Application Server

The JACC provider in WebSphere Application Server is implemented by both the client and the server pieces of the Tivoli Access Manager. The client piece of Tivoli Access Manager is embedded in WebSphere Application Server. The server piece is located on a separate installable CD that is shipped as part of the WebSphere Application Server Network Deployment (ND) package.

The JACC provider is not the default authorization. You must configure WebSphere Application Server to use the JACC provider.

JACC policy context handlers:

WebSphere Application Server supports all of the policy context handlers that are required by the Java Authorization Contract for Containers (JACC) specification. However, due to performance impacts, the Enterprise JavaBeans (EJB) arguments policy context handler is not activated unless it is specifically required by the provider. Performance impacts result if objects must be created for each arguments of each EJB method.

If the provider supports and requires this context handler, select the **Requires the EJB arguments policy context handler for access decisions** check box in the External JACC provider link under the Authorization providers panel or by using scripting. Any changes to this option are effective after the servers are restarted. By default this option is disabled. Disable this option when using Tivoli Access Manager as the JACC provider, because the argument values are not required for access decisions.

JACC policy context identifiers (ContextID) format:

A policy context identifier is defined as a unique string that represents a policy context. A policy context contains all of the security policy statements as defined by the Java Contract for Containers (JACC) specification that affect access to the resources in a Web or Enterprise JavaBeans (EJB) module.

During policy propagation to the JACC provider, a PolicyConfiguration object is created for each policy context. The object is populated with the policy statements, represented by the JACC permission objects that correspond to the context. The object is propagated to the JACC provider using the JACC specification APIs.

WebSphere Application Server makes the contextID unique by using the `href:cellName/appName/moduleName` string as the contextID format for the modules. The `href` part of the string indicates that a hierarchical name is passed as the context ID.

The `cellName` represents the name of the deployment manager cell or the base cell where the application is installed. After an application is installed in one cell (for example, in a base application server where the cell name is `base1`) and is added to a deployment manager cell whose name is `cell1` by using addNode, the context ID for the modules in the application contain `base1` (not `cell1`) as the cell name because the application is initially installed in `base1`.

The `appName` part of the string in the context ID represents the application name containing the module. The `moduleName` refers to the name of the module.

As an example, the context ID for the module `Increment.jar` file in an application named DefaultApplication that is installed in `cell1` is the `href:cell1/DefaultApplication/Increment.jar` file.

JACC policy propagation:

When an application is installed or deployed in WebSphere Application Server, the security policy information in the application is propagated to the provider when the configuration is saved. The context ID for the application is saved in its `application.xml` file, that is used for propagating the policy to the Java Authorization Contract for Containers (JACC) provider, and also for access decisions for Java 2 Platform, Enterprise Edition (J2EE) resources.

When an application is uninstalled, the security policy information in the application is removed from the provider when the configuration is saved.

If the provider implemented the RoleConfiguration interface, the security policy information that is propagated to the policy provider also contains the authorization table information. See “Interfaces that support JACC” on page 1380 for more information about this interface.

If an application does not contain security policy information, the PolicyConfiguration (and the RoleConfiguration, if implemented) objects do not contain any information. The existence of empty PolicyConfiguration and RoleConfiguration objects indicates that security policy information for the module does not exist.

After an application is installed, it can be updated without being uninstalled and reinstalled. For example, a new module can be added to an existing application, or an existing module can be modified. In this instance, the information in the impacted modules is propagated to the provider by default. A module is impacted when the deployment descriptor of the module is changed as part of the update. If the provider supports the RoleConfiguration interfaces, the entire authorization table for that application is propagated to the provider.

If the security information is not propagated to the provider during application updates, you can set the com.ibm.websphere.security.jacc.propagateonappupdate Java virtual machine (JVM) property to `false` in the deployment manager, in a Network Deployment environment, or the unmanaged base application server. If this property is set to `false`, any updates to an existing application in the server are not propagated to the provider. You also can set this property on a per-application basis using the custom properties of an application. The wsadmin tool can be used to set the custom property of an application. If this property is set at the application level, any updates to that application are not propagated to the

provider. If the update to an application is a full update, for example, a new application enterprise archive (EAR) file is used to replace the existing one, and the provider is refreshed with the entire application security policy information.

As mentioned earlier, the security policy information is propagated to the JACC provider during the save operation. The `SystemOut.log` file indicates the success or failure of the propagation to the provider. Check the log file after the installation to ensure that the propagation had no problems. If the propagation had any problems, access to the application fails when Tivoli Access Manager is used as the JACC provider.

If the security policy information for the application is successfully propagated to the provider, the audit statements with the message key SECJ0415I appear. However, if there was a problem propagating the security policy information to the provider (for example: network problems, JACC provider is not available), the `SystemOut.log` files contain the error message with the message keys SECJ0396E during install or SECJ0398E during modification. The installation of the application is not stopped due to a failure to propagate the security policy to the JACC provider. Also, in the case of failure, no exception or error messages appear during the save operation. When the problem causing this failure is fixed, run the **propagatePolicyToJaccProvider** tool to propagate the security policy information to the provider without reinstalling the application. For more information, see “Propagating security policy of installed applications to a JACC provider using wsadmin scripting” on page 1518.

JACC registration of the provider implementation classes:

The JACC specification states that providers can plug in their provider using the javax.security.jacc.policy.provider and the javax.security.jacc.PolicyConfigurationFactory.provider system properties.

The javax.security.jacc.policy.provider property is used to set the policy object of the provider, while the javax.security.jacc.PolicyConfigurationFactory.provider property is used to set the provider PolicyConfigurationFactory implementation.

Although both system properties are supported in WebSphere Application Server, it is highly recommended that you use the configuration model that is provided. You can set these values using either the JACC configuration panel (see “Authorizing access to J2EE resources using Tivoli Access Manager” on page 1364 for more information) or by using wsadmin scripting. One of the advantages of using the configuration model instead of the system properties is that the information is entered in one place at the cell level, and is propagated to all nodes during synchronization. Also, as part of the configuration model, additional properties can be entered, as described in the JACC configuration panel.

Role-based security with embedded Tivoli Access Manager:

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization model uses the concepts of roles and resources. An example is provided here.

	Methods		
Roles	**getBalance**	**deposit**	**closeAccount**
Teller	granted	granted	
Cashier	granted		
Supervisor			granted

In the example of the banking application that is conceptualized in the previous table, three roles are defined: teller, cashier, and supervisor. Permission to perform the getBalance, deposit, and closeAccount application methods are mapped to these roles. From the example, you can see that users assigned the role, Supervisor, can run the closeAccount method, whereas the other two roles are unable to run this method.

The term, principal, within WebSphere Application Sever security refers to a person or a process that performs activities. *Groups* are logical collections of principals that are configured in WebSphere Application Server to promote the ease of applying security. Roles can be mapped to principals, groups, or both. The entry that is invoked in the following table indicates that the principal or group can invoke any methods that are granted to that role.

	Roles		
Principal/Group	**Teller**	**Cashier**	**Supervisor**
TellerGroup	Invoke		
CashierGroup		Invoke	
SupervisorGroup			
Frank: A principal who is not a member of any of the previous groups		Invoke	Invoke

In the previous example, the principal Frank, can invoke the getBalance and the closeAccount methods, but cannot invoke the deposit method because this method is not granted either the Cashier or the Supervisor role.

At the time of application deployment, the Java Authorization Contract for Container (JACC) provider of Tivoli Access Manager populates the Tivoli Access Manager-protected object space with any security policy information that is contained in the application deployment descriptor. This security information is used to determine access whenever the WebSphere Application Server resource is requested.

By default, the Tivoli Access Manager access check is performed using the role name, the cell name, the application name, and the module name.

Tivoli Access Manager access control lists (ACLs) determine which application roles are assigned to a principal. ACLs are attached to the applications in the Tivoli Access Manager-protected object space at the time of application deployment.

Principal-to-role mappings are managed from the WebSphere Application Server administrative console and are never modified using Tivoli Access Manager. Direct updates to ACLs are performed for administrative security users only.

The following sequence of events occur:

1. During application deployment, policy information is sent to the JACC provider of Tivoli Access Manager . This policy information contains permission-to-role mappings and role-to-principal and role-to-group mapping information.
2. The JACC provider of Tivoli Access Manager converts the information into the required format, and passes this information to the Tivoli Access Manager policy server.
3. The policy server adds entries to the Tivoli Access Manager-protected object space to represent the roles that are defined for the application and the permission-to-role mappings. A permission is represented as a Tivoli Access Manager-protected object and the role that is granted to this object is attached as an extended attribute.

Tivoli Access Manager integration as the JACC provider:

Tivoli Access Manager uses the Java Authorization Contract for Container (JACC) model in WebSphere Application Server to perform access checks.

Tivoli Access Manager consists of the following components:

- Run time
- Client configuration

- Authorization table support
- Access check
- Authentication using the PDLoginModule module

Tivoli Access Manager run-time changes that are used to support JACC

For the run-time changes, Tivoli Access Manager implements the PolicyConfigurationFactory and the PolicyConfiguration interfaces, as required by JACC. During the application installation, the security policy information in the deployment descriptor and the authorization table information in the binding files are propagated to the Tivoli provider using these interfaces. The Tivoli provider stores the policy and the authorization table information in the Tivoli Access Manager policy server by calling the respective Tivoli Access Manager application programming interfaces (API).

Tivoli Access Manager also implements the RoleConfigurationFactory and the RoleConfiguration interfaces. These interfaces are used to ensure that the authorization table information is passed to the provider with the policy information. See “Interfaces that support JACC” on page 1380 for more information about these interfaces.

Tivoli Access Manager client configuration

To configure the Tivoli Access Manager client, you can use either the administrative console or wsadmin scripting. You can access the administrative console panels for the Tivoli Access Manager client configuration by clicking **Security > Secure administration, applications, and infrastructure > External authorization providers**. Under Related Items, click **External JACC provider**. The Tivoli client must be set up to use the Tivoli Access Manager JACC Provider.

For more information about how to configure the Tivoli Access Manager client, see “Tivoli Access Manager JACC provider configuration” on page 1371.

Authorization table support

Tivoli Access Manager uses the RoleConfiguration interface to ensure that the authorization table information is passed to the Tivoli Access Manager provider when the application is installed or deployed. When an application is deployed or edited, the set of users and groups for the user or group-to-role mapping are obtained from the Tivoli Access Manager server, which shares the same Lightweight Directory Access Protocol (LDAP) server as WebSphere Application Server. This sharing is accomplished by plugging into the application management users or groups-to-role administrative console panels. The management APIs are called to obtain users and groups rather than relying on the WebSphere Application Server-configured LDAP registry.

Access check

When WebSphere Application Server is configured to use the JACC provider for Tivoli Access Manager , it passes the information to Tivoli Access Manager to make the access decision. The Tivoli Access Manager policy implementation queries the local replica of the access control list (ACL) database for the access decision.

Authentication using the PDLoginModule module

The custom login module in WebSphere Application Server can do the authentication. This login module is plugged in before the WebSphere Application Server-provided login modules. The custom login modules can provide information that can be stored in the Subject. If the required information is stored, no additional registry calls are made to obtain that information.

As part of the JACC integration, the Tivoli Access Manager-provided PDLoginModule module is also used to plug into WebSphere Application Server for both Lightweight Third Party Authentication (LTPA) and Simple WebSphere Authentication Mechanism (SWAM) authentication. The PDLoginModule module is modified to authenticate with the user ID or password. The module is also used to fill in the required attributes in the Subject so that no registry calls are made by the login modules in WebSphere Application Server. The information that is placed in the Subject is available for the Tivoli Access Manager policy object to use for access checking.

Note: V6.0.x SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

Tivoli Access Manager security for WebSphere Application Server:

WebSphere Application Server provides embedded IBM Tivoli Access Manager client technology to secure your WebSphere Application Server-managed resources.

The benefits of using Tivoli Access Manager that are described here are only applicable when Tivoli Access Manager client code is used with the Tivoli Access Manager server:

- Robust container-based authorization
- Centralized policy management
- Management of common identities, user profiles, and authorization mechanisms
- Single-point security management for Java 2 Platform, Enterprise Edition (J2EE) compliant and non-compliant J2EE resources using the administrative console for Tivoli Access Manager Web Portal Manager
- No requirements for coding or deployment changes to applications
- Easy management of users, groups, and roles using the WebSphere Application Server administrative console

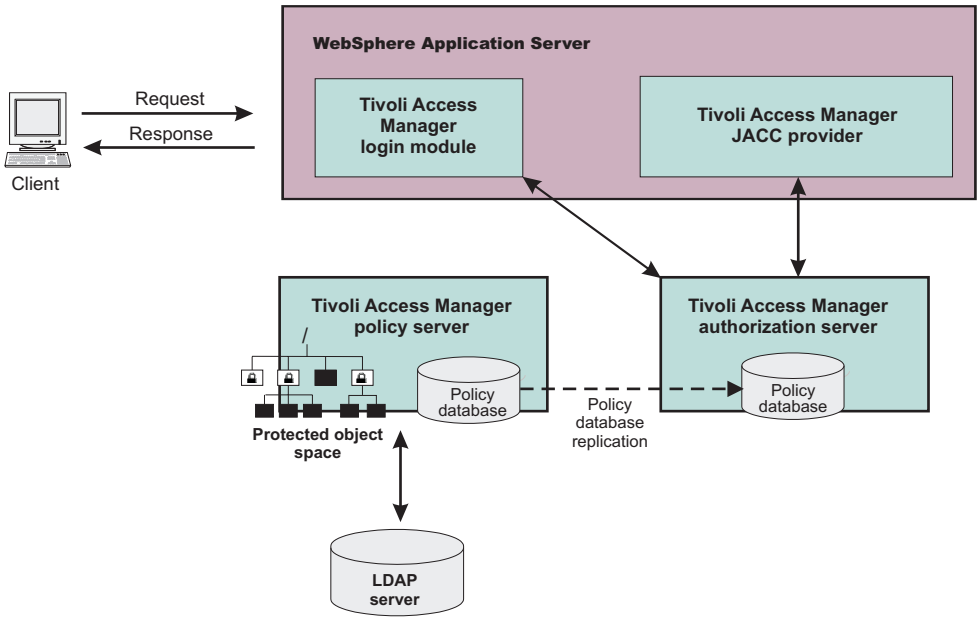
WebSphere Application Server supports the Java Authorization Contract for Containers (JACC) specification. JACC details the contract requirements for J2EE containers and authorization providers. With this contract, authorization providers can perform the access decisions for resources in J2EE Version 1.4 application servers such as WebSphere Application Server. The Tivoli Access Manager security utility that is embedded within WebSphere Application Server is JACC-compliant and is used to:

- Add security policy information when applications are deployed
- Authorize access to WebSphere Application Server-secured resources.

When applications are deployed, the embedded Tivoli Access Manger client takes any policy and or user and role information that is stored within the application deployment descriptor and stores it within the Tivoli Access Manager Policy Server.

The Tivoli Access Manager JACC provider is also called when a user requests access to a resource that is managed by WebSphere Application Server.

Embedded Tivoli Access Manager client architecture



The previous figure illustrates the following sequence of events:

1. Users that access protected resources are authenticated using the Tivoli Access Manager login module that is configured for use when the embedded Tivoli Access Manager client is enabled.
2. The WebSphere Application Server container uses information from the J2EE application deployment descriptor to determine the required role membership.
3. WebSphere Application Server uses the embedded Tivoli Access Manager client to request an authorization decision from the Tivoli Access Manager authorization server. Additional context information, when present, is also passed to the authorization server. This context information is comprised of the cell name, J2EE application name, and J2EE module name. If the Tivoli Access Manager policy database has policies that are specified for any of the context information, the authorization server uses this information to make the authorization decision.
4. The authorization server consults the permissions that are defined for the specified user within the Tivoli Access Manager-protected object space. The protected object space is part of the policy database.
5. The Tivoli Access Manager authorization server returns the access decision to the embedded Tivoli Access Manager client.
6. WebSphere Application Server either grants or denies access to the protected method or resource, based on the decision that is returned from the Tivoli Access Manager authorization server.

At its core, Tivoli Access Manager provides an authentication and authorization framework. You can learn more about Tivoli Access Manager, including the information that is necessary to make deployment decisions, by reviewing the product documentation. The following guides are available at the IBM Tivoli Access Manager for e-business information center:

- *IBM Tivoli Access Manager Base Installation Guide*

  This guide describes how to plan, install, and configure a Tivoli Access Manager secure domain. Using a series of easy installation scripts, you can quickly deploy a fully functional secure domain. These scripts are very useful when prototyping the deployment of a secure domain.

  To access this guide in the IBM Tivoli Access Manager for e-business information center, click **Access Manager for e-business > Base Information > Base Installation Guide**.

- *IBM Tivoli Access Manager Base Administration Guide*

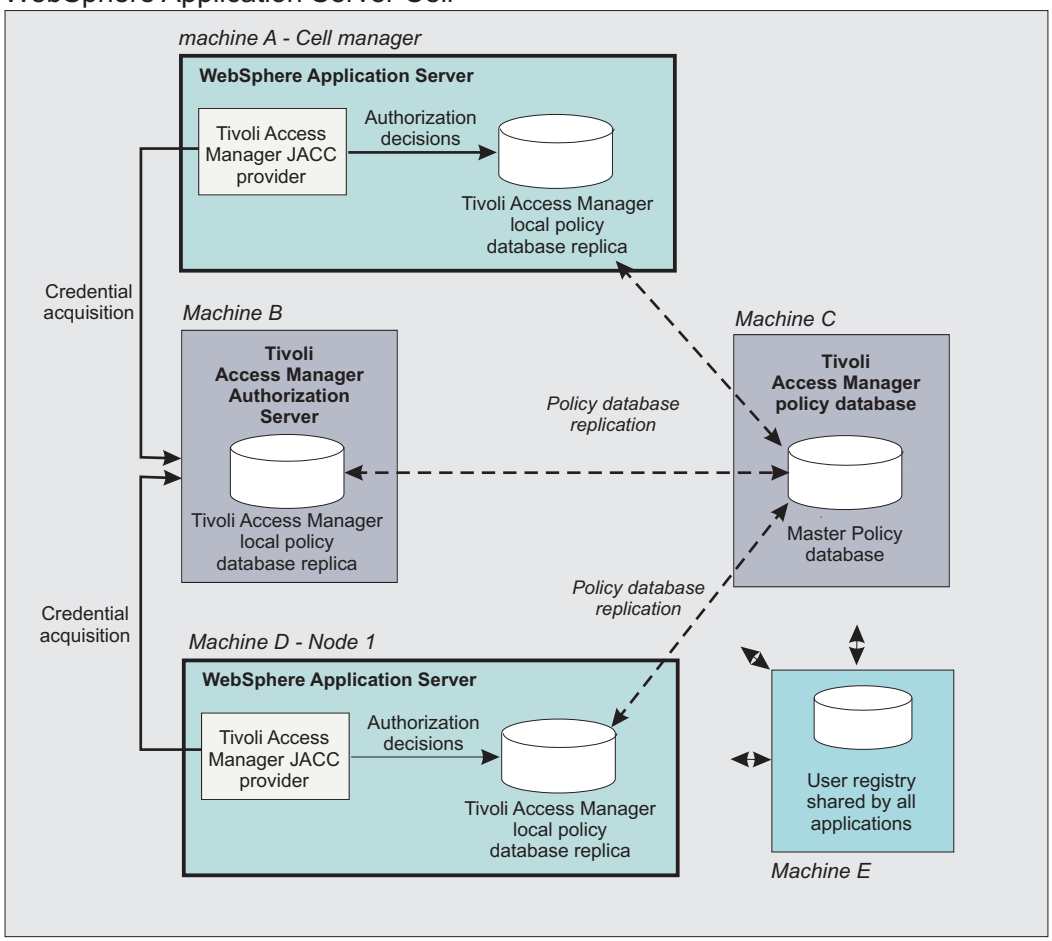
  This document presents an overview of the Tivoli Access Manager security model for managing protected resources. This guide describes how to configure the Tivoli Access Manager servers that

make access control decisions. In addition, detailed instructions describe how to perform important tasks, such as declaring security policies, defining protected object spaces, and administering user and group profiles.

To access this guide in the IBM Tivoli Access Manager for e-business information center, click **Access Manager for e-business > Base Information > Base Administration Guide**.

Tivoli Access Manager provides centralized administration of multiple servers.

WebSphere Application Server Cell



The previous figure is an example architecture showing WebSphere Application Servers secured by Tivoli Access Manager.

The participating WebSphere Application Servers use a local replica of the Tivoli Access Manager policy database to make authorization decisions for incoming requests. The local policy databases are replicas of the master policy database. The master policy database is installed as part of the Tivoli Access Manager installation. Having policy database replicas on each participating WebSphere Application Server node optimizes performance when making authorization decisions and provides failover capability.

Although the authorization server can also be installed on the same system as WebSphere Application Server, this configuration is not illustrated in the diagram.

All instances of Tivoli Access Manager and WebSphere Application Server in the example architecture share the Lightweight Directory Access Protocol (LDAP) user registry on Machine E.

The LDAP registries that are supported by WebSphere Application Server are also supported by Tivoli Access Manager.

It is possible to have separate WebSphere Application Server profiles on the same host that is configured for different Tivoli Access Manager servers. Such an architecture requires that the profiles are configured for separate Java Runtime Environments (JRE) and therefore you need multiple JREs installed on the same host.

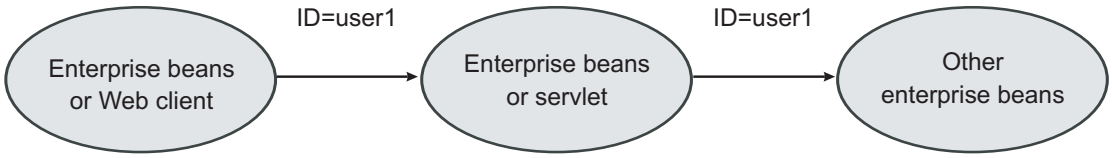
Delegations

Delegation is a process security identity propagation from a caller to a called object. As per the Java 2 Platform, Enterprise Edition (J2EE) specification, a servlet and enterprise beans can propagate either the client or remote user identity when invoking enterprise beans, or they can use another specified identity as indicated in the corresponding deployment descriptor.

The extension supports enterprise bean propagation to the server ID when invoking other entity beans. Three types of delegations are possible:

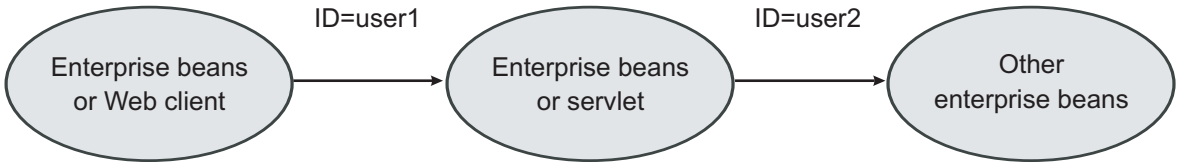
- Delegate (RunAs) client identity
- Delegate (RunAs) specified identity
- Delegate (RunAs) system identity

Delegate (RunAs) client identity



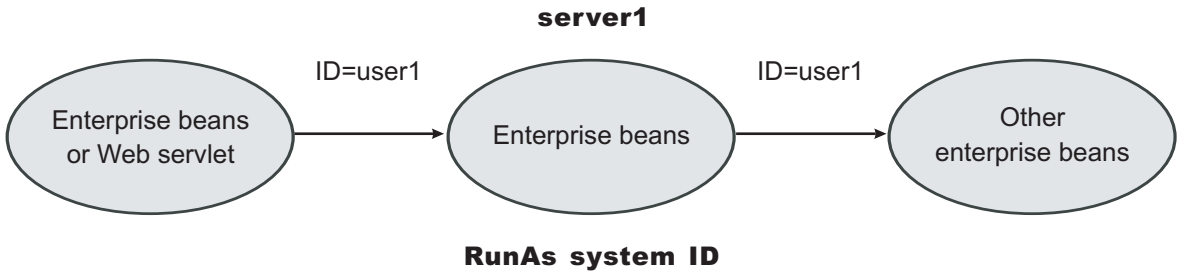
RunAs client ID

Delegate (RunAs) specified identity



Run As specified role mapped to user2

Delegate (RunAs) system identity



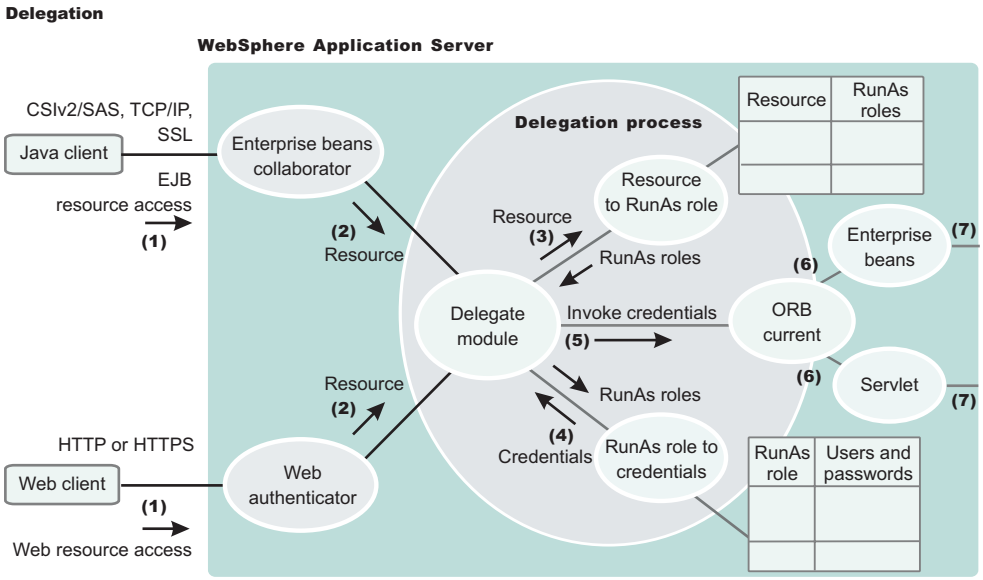
Note: The RunAs system identity delegation only works when server ID and password are used. When the internalServerId feature is used, it does not work because runAs with system identity is not supported. You must specify RunAs roles. When internalServerID is used, use the RunAsSpecified with a user ID and password that is mapped to the administrator role. See “Administrative roles and naming service authorization” on page 1334 for more information about internalServerId.

The EJB specification only supports delegation (RunAs) at the Enterprise JavaBeans (EJB) level. But an extension allows EJB method-level RunAs specification. With an EJB method level, the RunAs specification you can specify a different RunAs role for different methods within the same enterprise beans.

The RunAs specification is detailed in the deployment descriptor, which is the `ejb-jar.xml` file in the EJB module and the `web.xml` file in the Web module. The extension to the RunAs specification is included in the `ibm-ejb-jar-ext.xmi` file.

An IBM-specific binding file is available for each application that contains a mapping from the RunAs role to the user. This file is specified in the `ibm-application-bnd.xmi` file.

These specifications are read by the runtime during application startup. The following figure illustrates the delegation mechanism, as implemented in the WebSphere Application Server security model.



Delegation Process

Two tables help in the delegation process:
- Resource to RunAs role mapping table
- RunAs role to user ID and password mapping table

Use the Resource to RunAs role mapping table to get the role that is used by a servlet or by enterprise beans to propagate to the next enterprise beans call.

Use the RunAsRole to user ID and password mapping table to get the user ID that belongs to the RunAs role and its password.

Delegation is performed after successful authentication and authorization. During this process, the delegation module consults the Resource to RunAs role mapping table to get the RunAs role (3). The delegation module consults the RunAs role to user ID and password mapping table to get the user that belongs to the RunAs role (4). The user ID and password is used to create a new credential using the authentication module, which is not shown in the figure.

The resulting credential is stored in the Object Request Broker (ORB) Current as an invocation credential (5). Servlet and enterprise beans when invoking other enterprise beans pick up the invocation credential from the ORB Current (6) and call the next enterprise beans (7).

Programmatic login

Programmatic login is a type of form login that supports application presentation site-specific login forms for the purpose of authentication.

When enterprise bean client applications require the user to provide identifying information, the writer of the application must collect that information and authenticate the user. The work of the programmer can be broadly classified in terms of where the actual user authentication is performed:
- In a client program
- In a server program

Users of Web applications can receive prompts for authentication data in many ways. The `<login-config>` element in the Web application deployment descriptor file defines the mechanism that is used to collect this information. Programmers who want to customize login procedures, rather than relying on general purpose devices like a 401 dialog window in a browser, can use a form-based login to provide an application-specific HTML form for collecting login information.

No authentication occurs unless administrative security is enabled. If you want to use form-based login for Web applications, you must specify `FORM` in the auth-method tag of the `<login-config>` element in the deployment descriptor of each Web application.

Applications can present site-specific login forms by using the WebSphere Application Server form-login type. The Java 2 Platform, Enterprise Edition (J2EE) specification defines form login as one of the authentication methods for Web applications. WebSphere Application Server provides a form-logout mechanism.

Java Authentication and Authorization Service programmatic login

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is also mandated by the J2EE 1.4 Specification. JAAS is a collection of strategic authentication application programming interfaces (API) that replace the Common Object Request Broker Architecture (CORBA) programmatic login APIs. WebSphere Application Server provides some extensions to JAAS:

Before you begin developing with programmatic login APIs, consider the following points :

- For the pure Java client application or client container application, initialize the client Object Request Broker (ORB) security prior to performing a JAAS login. Do this by running the following code prior to the JAAS login:

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
// Perform an InitialContext and default lookup prior to logging
// in to initialize ORB security and for the bootstrap host/port
// to be determined for SecurityServer lookup. If you do not want
// to validate the userid/password during the JAAS login, disable
// the com.ibm.CORBA.validateBasicAuth property in the
// sas.client.props file.

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
     "corbaloc:iiop:myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
Object obj = initialContext.lookup("");
```

  For more information, see Example: Programmatic logins.
- For the pure Java client application or the client container application, make sure that the host name and the port number of the target Java Naming and Directory Interface (JNDI) bootstrap properties are specified properly. See the Developing applications that use CosNaming (CORBA Naming interface) section for details.
- If the application uses custom JAAS login configuration, make sure that the custom JAAS login configuration is properly defined. See the Configuring programmatic logins for Java Authentication and Authorization Service section for details.
- Some of the JAAS APIs are protected by Java 2 security permissions. If these APIs are used by application code, make sure that these permissions are added to the application `was.policy` file. See Adding the was.policy file to applications to the application, Using PolicyTool to edit policy files and Configuring the was.policy file sections for details. For more details of which APIs are protected by Java 2 Security permissions, check the IBM Developer Kit, Java Technology Edition; JAAS and the WebSphere Application Server public APIs documentation for more details. The following list contains the APIs that are used in the samples code provided in this documentation.
  - javax.security.auth.login.LoginContext constructors are protected by javax.security.auth.AuthPermission ″createLoginContext″.
  - javax.security.auth.Subject.doAs and com.ibm.websphere.security.auth.WSSubject.doAs are protected by javax.security.auth.AuthPermission ″doAs″.
  - javax.security.auth.Subject.doAsPrivileged and com.ibm.websphere.security.auth.WSSubject.doAsPrivileged are protected by javax.security.auth.AuthPermission ″doAsPrivileged″.
- com.ibm.websphere.security.auth.WSSubject: Due to a design oversight in JAAS Version 1.0, javax.security.auth.Subject.getSubject does not return the Subject associated with the running thread inside a java.security.AccessController.doPrivileged code block. This can present an inconsistent behavior that is problematic and causes an undesirable effort to work around. The com.ibm.websphere.security.auth.WSSubject API provides a workaround to associate the Subject to the running thread. The com.ibm.websphere.security.auth.WSSubject API extends the JAAS model to J2EE resources for authorization checks. The Subject that is associated with the running thread within com.ibm.websphere.security.auth.WSSubject.doAs or com.ibm.websphere.security.auth.WSSubject.doAsPrivileged code block is used for J2EE resources authorization checks.
- Administrative console support for defining new JAAS login configuration: You can configure JAAS login configuration in the administrative console and store it in the WebSphere Application Server configuration API. Applications can define new JAAS login configuration in the administrative console and the data is persisted in the configuration repository that is stored with the WebSphere Application

Server configuration API. However, WebSphere Application Server still supports the default JAAS login configuration format that is provided by the JAAS default implementation. If duplication login configurations are defined in both the WebSphere Application Server configuration API and the plain text file format, the login configuration in the WebSphere Application Server configuration API takes precedence. Advantages to define the login configuration in the WebSphere Application Server configuration API include:
  – Defining the JAAS login configuration using the administrative console.
  – Managing the JAAS login configuration centrally.
- JAAS login configurations for WebSphere Application Server: WebSphere Application Server provides JAAS login configurations for applications to perform programmatic authentication to the WebSphere Application Server security runtime. These JAAS login configurations for WebSphere Application Server perform authentication to the configured authentication mechanism, Simple WebSphere Authentication Mechanism (SWAM) or Lightweight Third-Party Authentication (LTPA), and user registry (Local OS, LDAP, or Custom) based on the authentication data supplied. The authenticated Subject from these JAAS login configurations contain the required principal and credentials that can be used by the WebSphere Application Server security runtime to perform authorization checks on J2EE role-based protected resources.

  Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

  Here are the JAAS login configurations that are provided by WebSphere Application Server:
  – *WSLogin JAAS login configuration*: A generic JAAS login configuration that a Java client, client container application, servlet, JSP file, enterprise bean, and so on, can use to perform authentication that is based on a user ID and password, or a token to the WebSphere Application Server security runtime. However, this configuration does not support the CallbackHandler handler that is specified in the client container deployment descriptor.
  – *ClientContainer JAAS login configuration*: This JAAS login configuration recognizes the CallbackHandler handler that is specified in the client container deployment descriptor. The login module of this login configuration uses the CallbackHandler handler in the client container deployment descriptor if one is specified, even if the application code specified one CallbackHandler handler in the login context. This is for client container application.
  – The Subjects that are authenticated with the previously mentioned JAAS login configurations contain a com.ibm.websphere.security.auth.WSPrincipal principal and a com.ibm.websphere.security.auth.WSCredential credential. If the authenticated Subject is passed to the com.ibm.websphere.security.auth.WSSubject.doAs method or the other doAs methods, the WebSphere Application Server security runtime can perform authorization checks on J2EE resources, based on the Subject com.ibm.websphere.security.auth.WSCredential credential.
- **Customer-defined JAAS login configurations**: You can define other JAAS login configurations. See Configuring programmatic logins for Java Authentication and Authorization Service for details. Use these login configurations to perform programmatic authentication to the custom authentication mechanism. However, the subjects from these customer-defined JAAS login configurations might not be used by the WebSphere Application Server security runtime to perform authorization checks if the subject does not contain the required principal and credentials.

Finding the root cause login exception from a JAAS login

If you get a LoginException exception after issuing the LoginContext.login API, you can find the root cause exception from the configured user registry. In the login modules, the registry exceptions are wrapped by a com.ibm.websphere.security.auth.WSLoginFailedException class. This exception has a getCause method with which you can pull out the exception that was wrapped after issuing the previous command.

You are not always guaranteed to get a WSLoginFailedException exception, but most of the exceptions that are generated from the user registry display here. The following example illustrates a LoginContext.login API with the associated catch block. Cast the WSLoginFailedException exception to com.ibm.websphere.security.auth.WSLoginFailedException class if you want to issue the getCause API.

The following determineCause example can be used for processing CustomUserRegistry exception types.

```
try
    {
         lc.login();
    }
    catch (LoginException le)
    {
 // drill down through the exceptions as they might cascade through the runtime
 Throwable root_exception = determineCause(le);

 // now you can use "root_exception" to compare to a particular exception type
 // for example, if you have implemented a CustomUserRegistry type, you would
  //  know what to look for here.
    }


/* Method used to drill down into the WSLoginFailedException to find the
"root cause" exception */

    public Throwable determineCause(Throwable e)
      {
    Throwable root_exception = e, temp_exception = null;

    // keep looping until there are no more embedded WSLoginFailedException or
    // WSSecurityException exceptions
         while (true)
    {
      if (e instanceof com.ibm.websphere.security.auth.WSLoginFailedException)
      {
       temp_exception = ((com.ibm.websphere.security.auth.WSLoginFailedException)
       e).getCause();
      }
      else if (e instanceof com.ibm.websphere.security.WSSecurityException)
      {
       temp_exception = ((com.ibm.websphere.security.WSSecurityException)
       e).getCause();
      }
      else if (e instanceof javax.naming.NamingException)
        // check for Ldap embedded exception
        {
          temp_exception = ((javax.naming.NamingException)e).getRootCause();
        }
      else if (e instanceof your_custom_exception_here)
      {
        // your custom processing here, if necessary
      }
      else
      {
        // this exception is not one of the types we are looking for,
        // lets return now, this is the root from the WebSphere
        //  Application Server perspective
        return root_exception;
      }
      if (temp_exception != null)
      {
        // we have an exception; go back and see if this has another
        // one embedded within it.
        root_exception = temp_exception;
        e = temp_exception;
        continue;
      }
      else
```

```
      {
        // we finally have the root exception from this call path, this
        // has to occur at some point
        return root_exception;
      }
    }
  }
```

Finding the root cause login exception from a Servlet filter

You can also receive the root cause exception from a servlet filter when addressing post-form login processing. This exception is useful because it shows the user what happened. You can issue the following API to obtain the root cause exception:

```
Throwable t = com.ibm.websphere.security.auth.WSSubject.getRootLoginException();
if (t != null)
         t = determineCause(t);
```

When you have the exception, you can run it through the previous determineCause example to get the native registry root cause.

Enabling root cause login exception propagation to pure Java clients

Currently, the root cause does not get propagated to a pure client for security reasons. However, you might want to propagate the root cause to a pure client in a trusted environment. If you want to enable root cause login exception propagation to a pure client, click **Security > Secure administration, applications, and infrastructure > Custom Properties** on the WebSphere Application Server Administrative Console and set the following property:

```
com.ibm.websphere.security.registry.propagateExceptionsToClient=true
```

Non-prompt programmatic login

WebSphere Application Server provides a non-prompt implementation of the javax.security.auth.callback.CallbackHandler interface, which is called com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl. Using this interface, an application can push authentication data to the WebSphere LoginModule instance to perform authentication. This capability is useful for server-side application code to authenticate an identity and to use that identity to invoke downstream J2EE resources.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl("user",
      "securityrealm", "securedpassword"));

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication data is "push" to the authentication mechanism
//   implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// maybe javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
```

```
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resource using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is a protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

You can use the com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl callback handler with a pure Java client, a client application container, enterprise bean, JavaServer Pages (JSP) files, servlet, or other Java 2 Platform, Enterprise Edition (J2EE) resources. See Example: Programmatic logins for more information about Object Request Broker (ORB) security initialization requirements in a pure Java client.

Note: The WSCallbackHandlerImpl callback handler is different depending on whether you use WebSphere Application Server security or Web services security. It is located in the `sas.jar` file for security, and in the `was-wssecurity.jar` file for Web services security.

User interface prompt programmatic login

WebSphere Application Server also provides a user interface implementation of the javax.security.auth.callback.CallbackHandler implementation to collect authentication data from a user through user interface login prompts. The com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl callback handler presents a user interface login panel to prompt users for authentication data.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by GUI login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// maybe javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
```

```
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is a protected enterprise bean
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Attention: Do not use the com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl callback handler for server-side resources like enterprise bean, servlet, JSP files, and so on. The user interface login prompt blocks the server for user input. This behavior is not good for a server process.

Stdin prompt programmatic login

WebSphere Application Server also provides a stdin implementation of the javax.security.auth.callback.CallbackHandler interface. The callback handler, com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl, prompts and collects authentication data from a user through the stdin prompt.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determines how authentication data is collected
// in this case, the authentication date is collected by stdin prompt
// and passed to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
          " + e.getMessage());
e.printStackTrace();

// maybe javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
```

```
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resource using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);
// where bankAccount is a protected enterprise bean
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
       + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Attention: Do not use the com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl callback handler for server-side resources like enterprise beans, servlets, JSP files, and so on. The input from the stdin prompt is not sent to the server environment. Most servers run in the background and do not have a console. However, if the server does have a console, the stdin prompt blocks the server for user input. This behavior is not good for a server process.

Authorizing access to J2EE resources using Tivoli Access Manager

The Java Authorization Contract for Containers (JACC) defines a contract between Java 2 Platform, Enterprise Edition (J2EE) containers and authorization providers. You can use the default authorization or an external JACC authorization provider. When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified.

JACC enables any third-party authorization providers to plug into a J2EE application server (such as WebSphere Application Server) to make the authorization decisions when a J2EE resource is accessed. By default, WebSphere Application Server implements the JACC provider by using Tivoli Access Manager as the external authorization provider.

Read the following articles for more detailed information about JACC before you attempt to configure WebSphere Application Server to use a JACC provider:

- “JACC support in WebSphere Application Server” on page 1344
- “JACC providers” on page 1346
- “Tivoli Access Manager integration as the JACC provider” on page 1350

Using the default authorization provider

You can extend the capabilities of WebSphere Application Server by plugging in your own authorization provider. You can use the default authorization or an external JACC authorization provider.

For an explanation of the administrative console panels that support these capabilities, see:

- Use the default authorization provider. It is recommended that you do not modify any settings on the authorization provider panels if you use the **Default authorization** option. For more information, see "External authorization provider settings."
- Use an external authorization provider. If you use the **External authorization using a JACC provider** option, the external providers must be based on the Java™ Authorization Contract for Containers (JACC) specification to handle the Java 2 Platform, Enterprise Edition (J2EE) authorization. By default, WebSphere Application Server enables you to configure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider as the default external JACC provider. For more information, see "External Java Authorization Contract for Containers provider settings" and "Tivoli Access Manager JACC provider settings" on page 1372.

External authorization provider settings:

Use this page to enable a Java Authorization Contract for Containers (JACC) provider for authorization decisions.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Click **External authorization providers**.

The application server provides a default authorization engine that performs all of the authorization decisions. In addition, the application server also supports an external authorization provider using the JACC specification to replace the default authorization engine for Java 2 Platform, Enterprise Edition (J2EE) applications.

JACC is part of the J2EE specification, which enables third-party security providers such as Tivoli Access Manager to plug into the application server and make authorization decisions.

Important: Unless you have an external JACC provider or want to use a JACC provider for Tivoli Access Manager that can handle J2EE authorizations based on JACC, and it is configured and set up to use with the application server, do not enable **External authorization using a JACC provider**.

Default authorization:

Use this option all the time unless you want an external security provider such as the Tivoli Access Manager to perform the authorization decision for J2EE applications that are based on the JACC specification.

Default: Enabled

External JACC provider: Use this link to configure the application server to use an external JACC provider. For example, to configure an external JACC provider, the policy class name and the policy configuration factory class name are required by the JACC specification.

The default settings that are contained in this link are used by Tivoli Access Manager for authorization decisions. If you intend to use another provider, modify the settings as appropriate.

External Java Authorization Contract for Containers provider settings:

Use this page to configure the application server to use an external Java Authorization Contract for Containers (JACC) provider. For example, the policy class name and the policy configuration factory class name are required by the JACC specification.

Use these settings when you have set up an external security provider that supports the JACC specification to work with the application server. The configuration process involves installing and configuring the provider server and configuring the client of the provider in the application server to communicate with the server. If the JACC provider is not enabled, these settings will be ignored.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Click **External authorization providers**.
3. Under Related items, click **External JACC provider**.

Use the default settings when you use Tivoli Access Manager as the JACC provider. Install and configure the Tivoli Access Manager server prior to using it with the application server. Use the Tivoli Access Manager properties link under Additional properties, and configure the Tivoli Access Manager client in the application server to use the Tivoli Access Manager server. If you intend to use another provider, modify the settings as appropriate.

Name:

Specifies the name that is used to identify the external JACC provider.

This field is required.

Data type: String

Description:

Provides an optional description for the provider.

Data type: String

Policy class name:

Specifies a fully qualified class name that represents the javax.security.jacc.policy.provider property as per the JACC specification. The class represents the provider-specific implementation of the java.security.Policy abstract methods.

The class file must reside in the class path of each application server process. This class is used during authorization decisions. The default class name is for Tivoli Access Manager implementation of the policy file.

This field is required. For information on enabling the JACC provider using this field, see the ″Enabling the JACC provider for Tivoli Access Manager″ article in the information center.

Data type: String
Default: com.tivoli.pd.as.jacc.TAMPolicy

Policy configuration factory class name:

Specifies a fully qualified class name that represents the javax.security.jacc.PolicyConfigurationFactory.provider property as per the JACC specification. The class represents the provider-specific implementation of the javax.security.jacc.PolicyConfigurationFactory abstract methods.

This class represents the provider-specific implementation of the PolicyConfigurationFactory abstract class. The class file must reside in the class path of each application server process. This class is used to propagate the security policy information to the JACC provider during the installation of the J2EE application. The default class name is for the Tivoli Access Manager implementation of the policy configuration factory class name.

This field is required.

Data type: String
Default: com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory

Role configuration factory class name:

Specifies a fully qualified class name that implements the com.ibm.wsspi.security.authorization.RoleConfigurationFactory interface.

The class file must reside in the class path of each application server process. When you implement this class, the authorization table information in the binding file is propagated to the provider during the installation of the J2EE application. The default class name is for the Tivoli Access Manager implementation of the role configuration factory class name.

This field is optional. For information on enabling the JACC provider using this field, see the ″Enabling the JACC provider for Tivoli Access Manager″ article in the information center.

Data type: String
Default: com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory

Provider initialization class name:

Specifies a fully qualified class name that implements the com.ibm.wsspi.security.authorization.InitializeJACCProvider interface.

The class file must reside in the class path of each application server process. When implemented, this class is called at the start and the stop of all the application server processes. You can use this class for any required initialization that is needed by the provider client code to communicate with the provider server. The properties that are entered in the custom properties link are passed to the provider when the process starts up. The default class name is for the Tivoli Access Manager implementation of the provider initialization class name.

This field is optional. For information on enabling the JACC provider using this field, see the ″Enabling the JACC provider for Tivoli Access Manager″ article in the information center.

Data type: String
Default: com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize

Requires the EJB arguments policy context handler for access decisions:

Specifies whether the JACC provider requires the EJBArgumentsPolicyContextHandler handler to make access decisions.

Because this option has an impact on performance, do not set it unless it is required by the provider. Normally, this handler is required only when the provider supports instance-based authorization. Tivoli Access Manager does not support this option for J2EE applications.

Default: Disabled

Supports dynamic module updates:

Specifies whether you can apply changes made to security policies of Web modules in a running application, dynamically without affecting the rest of the application.

If this option is enabled, the security policies of the added or modified Web modules are propagated to the JACC provider and only the affected Web modules are started.

If this option is disabled, then the security policies of the entire application are propagated to the JACC provider for any module-level changes. The entire application is restarted for the changes to take effect.

Typically, this option is enabled for an external JACC provider.

Default: Enabled

Custom properties:

Specifies the properties that are required by the provider.

These properties are propagated to the provider during the startup process when the provider initialization class name is initialized. If the provider does not implement the provider initialization class name as described previously, the properties are not used.

The Tivoli Access Manager implementation does not require that you enter any properties in this link.

Tivoli Access Manager properties:

Specifies properties that are required by the Tivoli Access Manager implementation.

These properties are used to set up the communication between the application server and the Tivoli Access Manager server. You must install and configure the Tivoli Access Manager server before entering these properties.

Enabling an external JACC provider

Use this topic to enable an external JACC provider using the administrative console.

The Java Authorization Contract for Containers (JACC) defines a contract between Java 2 Platform, Enterprise Edition (J2EE) containers and authorization providers. This contract enables any third-party authorization providers to plug into a J2EE 1.4 application server, such as WebSphere Application Server to make the authorization decisions when a J2EE resource is accessed.

1. From the WebSphere Application Server administrative console, click **Security > Secure administration, applications, and infrastructure > External authorization providers**.
2. Under Related items, click **External JACC provider**.
3. The fields are set for Tivoli Access Manager by default. If you do not plan to use Tivoli Access Manager as the JACC provider, replace these fields with the details for your own external JACC provider.
4. If any custom properties are required by the JACC provider, click **Custom properties** under Additional properties and enter the properties. When using the Tivoli Access Manager, use the **Tivoli Access Manager properties** link instead of the Custom properties link. For more information, see “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 1369.

5. On the External authorization providers panel, select the **External authorization using a JACC provider** option and click **OK**.
6. Complete the remaining steps to enable security. If you are using Tivoli Access Manager, you must select LDAP as the user registry and use the same LDAP server. For more information on configuring LDAP registries, see “Configuring Lightweight Directory Access Protocol user registries” on page 1088.
7. In a multinode environment, stop and start the deployment manager configuration.

   Issue the following commands:

   ```
   profile_root/bin/stopManager.bat
    -username user_name
    -password password

   profile_root/bin/startManager.bat
   ```
8. Restart all servers to make these changes effective.

Configuring the JACC provider for Tivoli Access Manager using the administrative console:

Use this task to configure Tivoli Access Manager as the Java Authorization Contract for Containers (JACC) provider using the administrative console.

Prior to completing the following steps, verify that you have previously created a security administrative user. For more information, see “Creating the security administrative user” on page 1370.

The following configuration is performed on the management server. When you click either **Apply** or **OK**, configuration information is checked for consistency, saved, and applied if successful.

To configure Tivoli Access Manager as the JACC provider using the administrative console, complete the following steps:

1. Start the WebSphere Application Server administrative console by clicking `http://yourhost.domain:port_number/ibm/console` after starting WebSphere Application Server. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password. This ID is typically the server user ID that is specified when you configure the user registry.
2. Click **Security > Secure administration, applications, and infrastructure > External authorization providers**.
3. Under General properties, select **External authorization using a JACC provider**.
4. Under Related items, click **External JACC provider**.
5. Under Additional properties, click **Tivoli Access Manager Properties**. The Tivoli Access Manager JACC provider configuration screen is displayed.
6. Enter the following information:

   Enable embedded Tivoli Access Manager
   Select this option to enable Tivoli Access Manager.

   Ignore errors during embedded Tivoli Access Manager disablement
   Select this option when you want to unconfigure the JACC provider. Do not select this option during configuration.

   Client listening port set
   WebSphere Application Server must listen using a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node or machine. More than one authorization server can be specified by separating the entries with commas. Specifying more than one authorization server at a time is useful for reasons of failover and performance. Enter the listening ports used by Tivoli Access Manager clients, separated by a comma. If a range of ports is specified, separate the lower and higher values by a colon (**:**) (for example, `7999, 9990:999`).

Policy server
Enter the name of the Tivoli Access Manager policy server and the connection port. Use the `policy_server:port` form. The policy communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7135.

Authorization servers
Enter the name of the Tivoli Access Manager authorization server. Use the `auth_server:port:priority` form. The authorization server communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7136. The priority value is determined by the order of the authorization server use (for example, `auth_server1:7136:1` and `auth_server2:7137:2`). A priority value of `1` is required when configuring against a single authorization server.

Administrator user name
Enter the Tivoli Access Manager administrator user name that was created when Tivoli Access Manager was configured; it is usually `sec_master`.

Administrator user password
Enter the Tivoli Access Manager administrator password.

User registry distinguished name suffix
Enter the distinguished name suffix for the user registry that is shared between Tivoli Access Manager and WebSphere Application Server, for example, `o=ibm, c=us`.

Security domain
You can create more than one security domain in Tivoli Access Manager, each with its own administrative user. Users, groups and other objects are created within a specific domain, and are not permitted to access resource in another domain. Enter the name of the Tivoli Access Manager security domain that is used to store WebSphere Application Server users and groups.

If a security domain is not established at the time of the Tivoli Access Manager configuration, leave the value as `Default`.

Administrator user distinguished name
Enter the full distinguished name of the WebSphere Application Server security administrator ID (for example, `cn=wasdmin, o=organization, c=country`). The ID name must match the Server user ID on the Lightweight Directory Access Protocol (LDAP) User Registry panel in the administrative console. To access the LDAP User Registry panel, click **Security > Secure administration, applications, and infrastructure**. Under **User account repository**, choose **Standalone LDAP registry** as the available realm definition. Then click **Configure**.

7. When all information is entered, click **OK** to save the configuration properties. The configuration parameters are checked for validity and the configuration is attempted at the host server or cell manager.

After you click **OK**, WebSphere Application Server completes the following actions:

- Validates the configuration parameters.
- Configures the host server or cell manager.

These processes might take some time depending on network traffic or the speed of your machine.

If the configuration is successful, the parameters are copied to all subordinate servers, including the node agents. To complete the embedded Tivoli Access Manager client configuration, you must restart all of the servers, including the host server, and enable WebSphere Application Server security.

Creating the security administrative user:

Enabling security requires the creation of a WebSphere Application Server administrative user. Use the Tivoli Access Manager command-line pdadmin utility to create the Tivoli Access Manager administrative user for WebSphere Application Server. This utility is available on the policy server host machine.

Follow these steps to use the pdadmin utility.

1. From a command line, start the pdadmin utility as the Tivoli Access Manager administrative user, sec_master:

   ```
   pdadmin -a sec_master -p sec_master_password
   ```

2. Create a WebSphere Application Server security user. For example, the following instructions create a new user, *wasadmin*. The command is entered as one continuous line:

   ```
   pdadmin> user create wasadmin cn=wasadmin,o=organization,
   c=country wasadmin wasadmin myPassword
   ```

   Substitute values for organization and country that are valid for your Lightweight Directory Access Protocol (LDAP) user registry.

3. Enable the account for the WebSphere Application Server security administrative user by issuing the following command:

   ```
   pdadmin> user modify wasadmin account-valid yes
   ```

Configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager. For more information, see “Tivoli Access Manager JACC provider configuration.”

Tivoli Access Manager JACC provider configuration:

You can configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager to deliver authentication and authorization protection for your applications or for authentication only. Most deployments that use the JACC provider for Tivoli Access Manager to configure Tivoli Access Manager provide both authentication and authorization functionality.

If you want Tivoli Access Manager to provide authentication, but leave authorization as part of WebSphere Application Server’s native security, add the com.tivoli.pd.as.amwas.DisableAddAuthorizationTableEntry=true property to the amwas.amjacc.template.properties file. The file is located in the *profile_root*/config/cells/*cell_name* directory.

After this property is set, perform the tasks for setting Tivoli Access Manager Security, as documented.

You can configure the JACC provider for Tivoli Access Manager using either the WebSphere Application Server administrative console or the **wsadmin** command-line utility.

- For details on configuring the JACC provider for Tivoli Access Manager using the administrative console, refer to “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 1369.
- For details on configuring the Tivoli Access Manager JACC provider using the **wsadmin** command line utility, refer to “Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility” on page 1519.

The JACC configuration files for Tivoli Access Manager that are common across multiple WebSphere Application Server profiles are created by default under the java/jre directory. When you install WebSphere Application Server, you are given permissions to read and write to the files in this directory.

This situation is not ideal because configuration of the JACC provider for Tivoli Access Manager fails in these situations. To avoid this situation, you can add the following property to the *profile_root*/config/cells/*cell_name*/amwas.amjacc.template.properties file: `com.tivoli.pd.as.jacc.CommonFileLocation=`*`new location`* where *new location* is a fully qualified directory name.

This property applies read and write permissions to the java/jre directory.

The **wsadmin** command is available to reconfigure the Java Authorization Contract for Containers (JACC) Tivoli Access Manager interface:

$AdminTask reconfigureTAM -interactive

This command effectively prompts you through the process of unconfiguring the interface and then reconfiguring it.

Tivoli Access Manager JACC provider settings:

Use this page to configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager.

Note: When a third-party authorization such as Tivoli Access Manager or SAF for z/OS is used, the information in the administrative console panel might not represent the data in the provider. Also, any changes to the panel might not be reflected in the provider automatically. Follow the provider's instructions to propagate any changes made to the provider.

To view the JACC provider settings for Tivoli Access Manager, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **External authorization providers**.
3. Under Related items, click **External JACC provider**.
4. Under Additional properties, click **Tivoli Access Manager Properties**.

Enable embedded Tivoli Access Manager:

Enables or disables the embedded Tivoli Access Manager client configuration.

Default:	Disabled
Range:	Enabled or Disabled

Note: If you want to disable Tivoli Access Manager as the JACC provider, clear this option and also select **Default authorization**.

Ignore errors during embedded Tivoli Access Manager disablement:

When selected, errors are ignored during disablement of the embedded Tivoli Access Manager client.

This option is applicable only when re-configuring an embedded Tivoli Access Manager client or disabling an embedded Tivoli Access Manager.

Default:	Disabled
Range:	Enabled or Disabled

Client listening port set:

Enter the ports that are used as listening ports by Tivoli Access Manager clients.

The application server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine, so a list of ports is required for

use by the processes. If you specify a range of ports, separate the lower and higher values by a colon (:). Single ports and port ranges are specified on separate lines. An example list might look like the following example:

```
7999
 9900:9999
```

Note: Each of the servants might need to open up a listener port.

Policy server:

Enter the name, fully-qualified domain name, or IP address of the Tivoli Access Manager policy server and the connection port.

Use the form *policy_server*:*port*. The policy server communication port was set at the time of the Tivoli Access Manager configuration. The default is 7135.

Authorization servers:

Enter the name, fully-qualified domain name, or IP address of the Tivoli Access Manager authorization server. Use the form, *auth_server*:*port*:*priority*.

The authorization server communication port is set at the time of Tivoli Access Manager configuration. The default is 7136. You can specify more than one authorization server by entering each server on a new line. Configuring more than one authorization server provides for failover. The priority value is the order of authorization server use. For example:

```
auth_server1.mycompany.com:7136:1
auth_server2.mycompany.com:7137:2
```

A priority of 1 is still required when configuring a single authorization server.

Administrator user name:

Enter the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This ID is usually, `sec_master`.

Administrator user password:

Enter the Tivoli Access Manager administration password for the user ID that is entered in the **Administrator user name** field.

User registry distinguished name suffix:

Enter the distinguished name suffix for the user registry to share between Tivoli Access Manager and the application server. For example: o=*organization*,c=*country*

Security domain:

Enter the name of the Tivoli Access Manager security domain that is used to store application server users and groups.

Specification of the Tivoli Access Manager domain is required because more than one security domain can be created in Tivoli Access Manager with its own administrative user. Users, groups, and other objects are created within a specific domain and are not permitted to access resources in another domain. If a security domain is not established at the time of Tivoli Access Manager configuration, leave the value as *Default*.

Default: Default

Administrator user distinguished name:

Enter the fully distinguished name of the security administrator ID for the application server. For example, cn=wasadmin,o=*organization*,c=*country*

JACC provider configuration properties for Tivoli Access Manager:

The JACC provider configuration properties detailed below may require configuration.

The Java property files are created in the *profile_root*/etc/tam directory.

Two properties files might require configuration:

- amwas.*node_name_server_name*.amjacc.properties contains properties that are used by the JACC provider of Tivoli Access Manager.
- amwas.*node_name_server_name*.pdjlog.properties contains logging properties that are created from the amwas.pdjlog.template.properties file for the specific node and server combination at the time of configuration.

Use amwas.*node_name_server_name*.amjacc.properties file to configure static role caching, dynamic role caching, object caching, and role-based policy framework properties.

Static role caching properties:

The static role cache holds role memberships that do not expire.

These properties are in the *profile_root*/etc/tam/amwas.*node_name_server_name*.amjacc.properties file.

The *profile_root* directory is the value of the profilePath parameter at profile creation time.

Enabling static role caching

com.tivoli.pd.as.cache.EnableStaticRoleCaching=true

Enables or disables static role caching. Static role caching is enabled by default.

Setting the static role cache

com.tivoli.pd.as.cache.StaticRoleCache=com.tivoli.pd.as.cache.StaticRoleCacheImpl

This property holds the implementation class of the static role cache. You do not need to change this property, although the opportunity exists to implement your own cache, if necessary.

Define static roles

com.tivoli.pd.as.cache.StaticRoleCache.Roles=Administrator,Operator,Monitor,Deployer

Defines the administration roles for WebSphere Application Server.

Tip: Enhance Application performance by adding the static roles: **CosNamingRead**, **CosNamingWrite**, **CosNamingCreate**, **CosNamingDelete**. These roles support for improved lookup performance within the application naming service.

Dynamic role caching properties:

The dynamic role cache holds role memberships that expire.

These properties are in the *profile_root*/etc/tam/amwas.*node_name_server_name*.amjacc.properties file.

The *profile_root* directory is the value of the profilePath parameter at profile creation time.

Enabling dynamic role caching

com.tivoli.pd.as.cache.EnableDynamicRoleCaching=true

Enables or disables dynamic role caching. Dynamic role caching is enabled by default.

Setting the dynamic role cache

com.tivoli.pd.as.cache.DynamicRoleCache=com.tivoli.pd.as.cache.DynamicRoleCacheImpl

This property holds the implementation class of the dynamic role cache. You do not need to change this property, although the opportunity exists to implement your own cache, if necessary.

Specifying the maximum number of users

com.tivoli.pd.as.cache.DynamicRoleCache.MaxUsers=100000

The maximum number of users that the cache supports before a cache cleanup is performed. The default number of users is 100000.

Specifying the number of cache tables

com.tivoli.pd.as.cache.DynamicRoleCache.NumBuckets=20

The number of tables that is used internally by the dynamic role cache. The default is 20. When a large number of threads use the cache, increase the value to tune and optimize cache performance.

Specifying the principal lifetime

com.tivoli.pd.as.cache.DynamicRoleCache.PrincipalLifeTime=10

The period of time in minutes that a principal entry is stored in the cache. The default time is 10 minutes. The term, *principal*, here refers to the Tivoli Access Manager credential that is returned from a unique Lightweight Directory Access Protocol user.

Specifying the role lifetime

com.tivoli.pd.as.cache.DynamicRoleCache.RoleLifetime=20

The period of time in seconds that a role is stored in the role list for a user before it is discarded. The default is 20 seconds.

Object caching properties:

The object cache is used to cache all Tivoli Access Manager objects, including their extended attributes. This bypasses the need to query the Tivoli Access Manager authorization server for each resource request.

These properties are in the *profile_root*/etc/tam/amwas.*node_name_server_name*.amjacc.properties file.

The *profile_root* directory is the value of the profilePath parameter when the profile is created.

Enabling object caching

com.tivoli.pd.as.cache.EnableObjectCaching=true

This property enables or disables object caching. The default value is true.

Setting the object cache

com.tivoli.pd.as.cache.ObjectCache=com.tivoli.pd.as.cache.ObjectCacheImpl

This property is the class used to perform object caching. You can implement your own object cache if required. This can be done by implementing the *com.tivoli.pd.as.cache.IObjectCache* interface. The default is *com.tivoli.pd.as.cache.ObjectCacheImpl*.

Setting the number of cache buckets

com.tivoli.pd.as.cache.ObjectCache.NumBuckets=20

This property specifies the number of buckets used to store object cache entries in the underlying hash table. The default is 20.

Setting the number of cache bucket entries

com.tivoli.pd.as.cache.ObjectCache.MaxResources=10000

This property specifies the total number of entries for all buckets in the cache. This figure, divided by NumBuckets determines the maximum size of each bucket. The default is 10000.

Setting the resource lifetime

com.tivoli.pd.as.cache.ObjectCache.ResourceLifeTime=20

This property specifies the length of time in minutes that objects are kept in the object cache. The default is 20.

These object cache properties cannot be changed after configuration. If any require changing, it should be done before configuration of the nodes in the cell. Changes need to be made in the template properties file before any configuration actions are performed. Properties changed after configuration might cause access decisions to fail.

Role-based policy framework properties:

Although it is very unlikely that you will need to change these properties, use this file to reference supported properties within the role-based policy framework.

The role-based policy framework parameters are located in the Java Authorization Contract for Containers (JACC) configuration file and in the authorization configuration file. These parameters are set at the time of JACC provider configuration and authorization server configuration. The role-based policy framework settings for the authorization table and the JACC provider can be modified separately for each WebSphere Application Server instance. The amwas.*node_server*.authztable.properties configuration file is generated from the authorization table. The configuration file is generated from the JACC provider, amwas.*node_name_server_name*.amjacc.properties. Both files are stored on the WebSphere Application Server *profile_root*/etc/tam directory. It is unlikely that you need to change these properties, but these properties are described here for reference:

Supported properties include:

com.tivoli.pd.as.rbpf.AMAction=i
This property is used to signify that a user is granted access to a role. This value is added to a Tivoli Access Manager access control list (ACL) and places invoke access on roles for users and groups.

com.tivoli.pd.as.rbpf.AMActionGroup=WebAppServer
This property sets the Tivoli Access Manager action group that serves as a container for the action that is specified by the com.tivoli.pd.as.rbpf.AMAction property. The permission set in the com.tivoli.pd.as.rbpf.AMAction property goes into this action group.

com.tivoli.pd.as.rbpf.PosRoot=WebAppServer
This property is used to determine where roles are stored in the protected object space.

com.tivoli.pd.as.rbpf.ProductId=deployedResources
This property specifies the location under the root location that is specified in the posroot property to separate other products in the protected object space. Embedded Tivoli Access Manager objects are found in the /WebAppServer/deployedResources directory. The default value is deployedResources.

com.tivoli.pd.as.rbpf.ResourceContainerName=Resources
This property specifies the Tivoli Access Manager object space container name for the protected resources. The default location is the /WebAppServer/deployedResources/Resources directory.

com.tivoli.pd.as.rbpf.RoleContainerName=Roles
This property specifies the Tivoli Access Manager protected object space container name for the security roles. The default location is the /WebAppServer/deployedResources/Roles directory.

The previous settings cannot be changed after configuration. Make changes in the template properties file before any configuration actions are performed. Properties that are changed after configuration will cause access decisions to fail.

System-dependent configuration properties:

Do not change these system-dependent configuration properties. These properties are included in this article for reference only.

These properties are in the *app_server_root*/etcamwas.*node_name_server_name*.amjacc.properties file.

The *profile_root* variable is the value of the profilePath parameter when the profile is created.

The supported arguments include:

com.tivoli.pd.as.rbpf.AmasSession.CfgURL=file/:$WAS_HOME/profiles/profile_name/etc/tam/amwas.node_server.pdperm.properties
This entry is generated by the Java Authorization Contract for Containers (JACC) provider configuration. This argument specifies the location of the file that contains information about the JACC provider of Tivoli Access Manager. Do not change this entry or the properties in the amwas.node_server.pdperm.properties file.

com.tivoli.pd.as.rbpf.AmasSession.LoggingURL=file/:$WAS_HOME/profiles/profile_name/etc/tam/amwas.node_server.pdjlog.properties
This entry contains the location of the logging configuration file for the JACC provider of Tivoli Access Manager. The referenced file is generated by the JACC provider of Tivoli Access Manager configuration. Do not change this entry.

Administering security users and roles with Tivoli Access Manager:

Use these steps to manage user-to-role mappings and user-to-group mappings for applications.

User-to-role mapping and user-to-group mapping for the JACC provider of Tivoli Access Manager are performed using the WebSphere Application Server administrative console.

1. Click **Applications > Enterprise applications >** *application_name*.
2. Under Additional properties, click **Security role to user/group mapping**. The user and groups management screen is displayed.
3. Select the role that requires user or group management and use **Lookup users** or **Lookup groups** to manage the users or groups for the selected role. The native role mapping uses the MapRolesToUsers administrative task. If you are using Tivoli Access Manager, use the TAMMapRolesToUsers administrative task instead. The syntax and options for the Tivoli version are the same as those used in the native version. For more information, see csec_role_based_sec.dita and tsec_use_TAM_groups.dita.

Configuring Tivoli Access Manager groups:

Use these steps to configure the WebSphere Application Server administrative console to add objects of the accessGroup class to the list of object classes that represent user registry groups.

You can use the WebSphere Application Server administrative console to specify security policies for applications that run in the WebSphere Application Server environment. You can also use the WebSphere Application Server administrative console specify security policies for other Web resources, based on the entities that are stored in the user registry.

Tivoli Access Manager adds the accessGroup object class to the registry. Tivoli Access Manager administrators can use the pdadmin utility, which is available only on the policy server host in the PD.RTE fileset, to create new groups. These new groups are added to the registry as the accessGroup object class.

The WebSphere Application Server administrative console is not configured by default to recognize objects of the accessGroup class as user registry groups. You can configure the WebSphere Application Server administrative console to add this object class to the list of object classes that represent user registry groups. To do this configuration, complete the following instructions:

1. From the WebSphere Application Server administrative console, access the advanced settings for configuring security by clicking **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry**, and click **Configure**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
4. Modify the **Group Filter** field. Add the following entry: `(objectclass=accessGroup)`

   The Group Filter field looks like the following example:

   ```
   (&(cn=%w)(|(objectclass=groupOfNames)
   (objectclass=groupOfUniqueNames)(objectclass=accessGroup)))
   ```
5. Modify the **Group Member ID Map** field. Add the following entry: `accessGroup:member` The Group Member ID Map field looks like the following example:

   ```
   groupOfNames:member;groupOfUniqueNames:uniqueMember;
   accessGroup:member
   ```
6. Stop and restart WebSphere Application Server.

Configuring additional authorization servers:

Tivoli Access Manager secure domains can contain more than one authorization server. Having multiple authorization servers is useful for providing a failover capability as well as improving performance when the volume of access requests is large.

1. Refer to the *Tivoli Access Manager Base Administration Guide* for details on installing and configuring authorization servers. This document is available in the IBM Tivoli Access Manager for e-business information center.
2. Re-configure the Java Authorization Contract for Containers (JACC) provider using the `$AdminTask reconfigureTAM interactive` **wsadmin** command. Enter all new and existing options.

Logging Tivoli Access Manager security:

Use this topic to enable the trace specification to indicate tracing at the required level.

The Java Authorization Contract for Containers (JACC) for Tivoli Access Manager provider messages are logged to the configured trace output location, and messages are written to standard out `SystemOut.log` file. When trace is enabled, all logging, both trace and messaging, is sent to the `trace.log` file.

1. The `amwas.node_server.pdjlog.properties` file must be updated and the **isLogging** attribute set to *true* for the required component. For example, to enable tracing for the JACC provider for Tivoli Access Manager, set the following line to *true*:
   `amwas.node_server.pdjlog.properties:baseGroup.AMWASWebTraceLogger.isLogging=true`
2. Enable tracing for the JACC provider of Tivoli Access Manager components in the WebSphere Application Server administrative console by completing the following steps:
   a. Click **Troubleshooting** > **Logs and Trace** > *server_name*.
   b. Under Logs and Trace tasks, click **Diagnostic trace**.
   c. Select the **Enable Log** option.
   d. Click **Apply**.
   e. Click **Troubleshooting** > **Logs and Trace** > *server_name*.
   f. Click **Change Log Detail Levels**.
   g. Click **Components**. Tracing for all components can be enabled using the **com.tivoli.pd.as.*** command. Tracing for separate components can be enabled using the following commands:
      - **com.tivoli.pd.as.rbpf.*** for role-based policy framework tracing
      - **com.tivoli.pd.as.jacc.*** for JACC provider tracing
      - **com.tivoli.pd.as.pdwas.*** for the authorization table
      - **com.tivoli.pd.as.cfg.*** for configuration
      - **com.tivoli.pd.as.cache.*** for caching

      For more information, see utrb_loglevel.dita.
   h. Click **Apply**.

The trace specification now indicates that tracing is enabled at the required level. Save the configuration and restart the server for the changes to take effect.

Tivoli Access Manager loggers:

The Java Authorization Contract for Containers (JACC) for Tivoli Access Manager provider messages are logged to the configured trace output location, and messages are written to standard out `SystemOut.log` file. When trace is enabled, all logging, both trace and messaging, is sent to the `trace.log` file.

The JACC provider for Tivoli Access Manager uses the JLog logging framework as does the Java runtime environment for Tivoli Access Manager. You can enable tracing and messaging selectively for specific JACC provider for Tivoli Access Manager components.

Tracing and message logging for the JACC provider for Tivoli Access Manager are configured in the `amwas.node_server.pdjlog.properties` properties file, which is located in the `profile_root/etc/tam`

directory. This file contains logging properties from the amwas.pdjlog.template.properties template file for the specific node and server combination at the time of JACC provider for Tivoli Access Manager configuration.

The contents of this file let the user control:

- Whether tracing is enabled or disabled for the JACC provider of Tivoli Access Manager components.
- Whether message logging is enabled or disabled for the JACC provider of Tivoli Access Manager components.

The amwas.*node_server*.pdjlog.properties file defines several loggers, each of which is associated with one JACC provider of Tivoli Access Manager component. These loggers include:

Logger Name	Description
AmasRBPFTraceLogger AmasRBPFMessageLogger	Logs messages and trace for the role-based policy framework. This underlying framework is used by embedded Tivoli Access Manager to make access decisions.
AmasCacheTraceLogger AmasCacheMessageLogger	Logs messages and trace for the policy caches that are used by the role-based policy framework.
AMWASWebTraceLogger AMWASWebMessageLogger	Logs messages and trace for the WebSphere Application Server authorization plug-in.
AMWASConfigTraceLogger AMWASConfigMessageLogger	Logs messages and trace for the configuration actions of the JACC provider for Tivoli Access Manager .
JACCTraceLogger JACCMessageLogger	Logs messages and trace for the JACC provider activity of Tivoli Access Manager .

Note: Tracing can have a significant impact on system performance. Enable tracing only when diagnosing the cause of a problem.

The implementation of these loggers routes messages to the WebSphere Application Server logging sub-system. All messages are written to the WebSphere Application Server trace.log file.

For each logger, the amwas.*node_server*.pdjlog.properties file defines an isLogging attribute which, when set to *true*, enables logging for the specific component. A value of *false* disables logging for that component.

The amwas.node_server.pdjlog.properties file defines the parent loggers MessageLogger and TraceLogger that also have an isLogging attribute. If the child loggers do not specify this isLogging attribute, they inherit the value of their respective parent. When the JACC provider for Tivoli Access Manager is enabled, the isLogging attribute is set to *true* for the MessageLogger and set to*false* for the TraceLogger logger. Message logging is enabled for all components and tracing is disabled for all components, by default.

To turn on tracing for a JACC provider component, see Logging Tivoli Access Manager security .

Interfaces that support JACC:

WebSphere Application Server provides the RoleConfigurationFactory and the RoleConfiguration interfaces, which are similar to PolicyConfigurationFactory and PolicyConfiguration interfaces so the information that is stored in the bindings file can be propagated to the provider during installation. The implementation of these interfaces is optional.

RoleConfiguration interface

Use the RoleConfiguration interface to propagate the authorization information to the provider. This interface is similar to the PolicyConfiguration interface that is found in Java Authorization Contact for Containers (JACC).

```
RoleConfiguration
      - com.ibm.wsspi.security.authorization.RoleConfiguration

/**
 * This interface is used to propagate the authorization table information
 * in the binding file during application installation. Implementation of this interface is
 * optional. When a JACC provider implements this interface during an application, both
 * the policy and the authorization table information are propagated to the provider.
* If this is not implemented, only the policy information is propagated as per
 * the JACC specification.
 * @ibm-spi
 * @ibm-support-class-A1
 */


public interface RoleConfiguration

/**
 * Add the users to the role in RoleConfiguration.
 * The role is created, if it does not exist in RoleConfiguration.
 * @param role the role name.
 * @param users the list of the user names.
 * @exception RoleConfigurationException if the users cannot be added.
 */
 public void addUsersToRole(String role, List users)
 throws RoleConfigurationException

/**
 * Remove the users to the role in RoleConfiguration.
 * @param role the role name.
 * @param users the list of the user names.
 * @exception RoleConfigurationException if the users cannot be removed.
 */
 public void removeUsersFromRole(String role, List users)
 throws RoleConfigurationException

/**
 * Add the groups to the role in RoleConfiguration.
 * The role is created if it does not exist in RoleConfiguration.
 * @param role the role name.
 * @param groups the list of the group names.
 * @exception RoleConfigurationException if the groups cannot be added.
 */
 public void addGroupsToRole(String role, List groups)
 throws RoleConfigurationException

/**
 * Remove the groups to the role in RoleConfiguration.
 * @param role the role name.
 * @param groups the list of the group names.
 * @exception RoleConfigurationException if the groups cannot be removed.
 */
 public void removeGroupsFromRole( String role, List groups)
 throws RoleConfigurationException

/**
 * Add the everyone to the role in RoleConfiguration.
 * The role is created if it does not exist in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the everyone cannot be added.
 */
 public void addEveryoneToRole(String role)
 throws RoleConfigurationException

/**
 * Remove the everyone to the role in RoleConfiguration.
 * @param role the role name.
```

```
 * @exception RoleConfigurationException if the everyone cannot be removed.
 */
 public void removeEveryoneFromRole( String role)
 throws RoleConfigurationException
/**
 * Add the all authenticated users to the role in RoleConfiguration.
 * The role is created if it does not exist in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the authentication users cannot
 *  be added.
 */
 public void addAuthenticatedUsersToRole(String role)
 throws RoleConfigurationException
/**
 * Remove the all authenticated users to the role in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the authentication users cannot
 * be removed.
 */
 public void removeAuthenticatedUsersFromRole( String role)
 throws RoleConfigurationException
/**
 * This commits the changes in Roleconfiguration.
 * @exception RoleConfigurationException if the changes cannot be
 * committed.
 */
 public void commit( )
 throws RoleConfigurationException
/**
 * This deletes the RoleConfiguration from the RoleConfiguration Factory.
 * @exception RoleConfigurationException if the RoleConfiguration cannot
 * be deleted.
 */
 public void delete( )
 throws RoleConfigurationException
/**
 * This returns the contextID of the RoleConfiguration.
 * @exception RoleConfigurationException if the contextID cannot be
 * obtained.
 */
 public String getContextID( )
 throws RoleConfigurationException
```

RoleConfigurationFactory interface

The RoleConfigurationFactory interface is similar to the PolicyConfigurationFactory interface that is introduced by JACC, and is used to obtain RoleConfiguration objects based on the contextID IDs.

```
RoleConfigurationFactory
 - com.ibm.wsspi.security.authorization.RoleConfigurationFactory

/**
 * This interface is used to instantiate the com.ibm.wsspi.security.authorization.RoleConfiguration
 * objects based on the context identifier similar to the policy context identifier.
 * Implementation of this interface is required only if the RoleConfiguration interface is implemented.
 *
 * @ibm-spi
 * @ibm-support-class-A1
 */

public interface RoleConfigurationFactory
/**
 * This gets a RoleConfiguration with contextID from the
 * RoleConfigurationfactory. If the RoleConfiguration does not exist
 * for the contextID in the RoleConfigurationFactory, a new
```

```
 * RoleConfiguration with contextID is created in the
 * RoleConfigurationFactory. The contextID is similar to
 * PolicyContextID, but it does not contain the module name.
 * If remove is true, the old RoleConfiguration is removed and a new
 * RoleConfiguration is created, and returns with the contextID.
 * @return the RoleConfiguration object for this contextID
 * @param contextID the context ID of RoleConfiguration
 * @param remove true or false
 * @exception RoleConfigurationException if RoleConfiguration
 * cannot be obtained.
 **/
public abstract com.ibm.ws.security.policy.RoleConfiguration
        getRoleConfiguration(String contextID, boolean remove)
      throws RoleConfigurationException
```

InitializeJACCProvider provider

When implemented by the provider, this interface is called by every process where the JACC provider can be used for authorization. All additional properties that are entered during the authorization check are passed to the provider. For example, the provider can use this information to initialize client code to communicate with their server or repository. The cleanup method is called during server shutdown to clean up the configuration.

Declaration

public interface **InitializeJACCProvider**

Description

This interface has two methods. The JACC provider can implement the interface, and WebSphere Application Server calls it to initialize the JACC provider. The name of the implementation class is obtained from the value of the initializeJACCProviderClassName system property.

This class must reside in a Java archive (JAR) file on the class path of each server that uses this provider.

```
InitializeJACCProvider
   - com.ibm.wsspi.security.authorization.InitializeJACCProvider

  /**
   * Initializes the JACC provider
      * @return 0 for success.
   * @param props the custom properties that are included for this provider will
   * pass to the implementation class.
   * @exception Exception for any problems encountered.
   **/
  public int initialize(java.util.Properties props)
  throws Exception

  /**
   * This method is for the JACC provider cleanup and will be called during a process stop.
   **/
  public void cleanup()
```

Enabling the JACC provider for Tivoli Access Manager:

The Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager is configured by default. Use this topic to enable the JACC provider for Tivoli Access Manager.

Restriction: Do not perform this task if you are configuring the JACC provider for Tivoli Access Manager to supply authentication services only. Only perform this task for installations that require both Tivoli Access Manager authentication and authorization protection.

The JACC provider for Tivoli Access Manager is configured by default. The following list shows the JACC provider configuration settings for Tivoli Access Manager:

Field	Value
Name	Tivoli Access Manager
Description	This field is optional and used as a reference.
J2EE policy class name	`com.tivoli.pd.as.jacc.TAMPolicy`
Policy configuration factory class name	`com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory`
Role configuration factory class name	`com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory`
JACC provider initialization class name	`com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize`
Requires the EJB arguments policy context handler for access decisions	false
Supports dynamic module updates	true

To enable the JACC provider for Tivoli Access Manager, use the previous settings and complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure > External authorization providers**.
2. Select the **External authorization using a JACC provider** option, then click **Apply**.
3. Under Related Items, click **External JACC provider**. The JACC provider settings for Tivoli Access Manager are displayed.
4. Verify that the correct settings are present to work with your Tivoli Access Manager configuration. For more information, see “External Java Authorization Contract for Containers provider settings” on page 1365.
5. Under Additional properties, click **Tivoli Access Manager properties**.
6. Click the **Enable embedded Tivoli Access Manager** option and verify that the correct Tivoli Access Manager server and WebSphere Application Server settings exist. For more information, see “Tivoli Access Manager JACC provider settings” on page 1372.
7. Click **OK**.
8. Save the settings by clicking **Save** at the top of the page.
9. Log out of the WebSphere Application Server administrative console.
10. Restart WebSphere Application Server. The security configuration is now replicated to managed servers and node agents. These other servers within a cell also require restarting before the security changes take effect.

Enabling embedded Tivoli Access Manager:

Embedded Tivoli Access Manager is not enabled by default, and you need to configure it for use.

Enabling Tivoli Access Manager security within WebSphere Application Server requires:

- A supported Lightweight Directory Access Protocol (LDAP) installed somewhere on your network. This user registry contains the user and group information for both Tivoli Access Manager and WebSphere Application Server.
- Tivoli Access Manager server exists and is configured to use the user registry. For details on the installation and configuration of Tivoli Access Manager, refer to the IBM Tivoli Access Manager for e-business information center.

  Note: WebSphere Application Server contains an embedded client for Tivoli Access Manager. To use Tivoli Access Manager, you must also configure the Tivoli Access Manager server.

However, the server version must be the same version or later as the client version. For information on the supported version of Tivoli Access Manager, see WebSphere Application Server - Supported Prerequisites.

- WebSphere Application Server is installed either in a single server model or as WebSphere Application Server Network Deployment.
- When administrative security is configured with a Federal Information Processing Standard (FIPS) provider, the Tivoli Access Manager server must be configured for FIPS as well

Complete the following steps to enable embedded Tivoli Access Manager security:

1. Create the security administrative user.
   For more information, see the *Securing applications and their environment* PDF.
2. Configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager .
   For more information, see the *Securing applications and their environment* PDF.
3. Enable WebSphere Application Server security. When you are using Tivoli Access Manager you must configure LDAP as the user registry.
   For more information, see the *Securing applications and their environment* PDF.
4. Enable the JACC provider for Tivoli Access Manager.
   For more information, see the *Securing applications and their environment* PDF.

Disabling embedded Tivoli Access Manager client:

To unconfigure Tivoli Access Manager Security in WebSphere Application Server, you can use either the wsadmin command-line utility or the WebSphere Application Server administrative console.

- For details on unconfiguring the embedded Tivoli Access Manager client using the WebSphere Application Server administrative console, refer to “Disabling embedded Tivoli Access Manager client using the administrative console.”
- For details on unconfiguring the embedded Tivoli Access Manager client using the wsadmin command line utility, refer to “Disabling embedded Tivoli Access Manager client using wsadmin” on page 1521.

Disabling embedded Tivoli Access Manager client using the administrative console:

To unconfigure the JACC provider for Tivoli Access Manager, you can use the WebSphere Application Server administrative console.

1. Click **Security > Secure administration, applications, and infrastructure > External authorization providers**.
2. Under Related items, click **External JACC provider**.
3. Under Additional properties, click **Tivoli Access Manager Properties**. The configuration screen for the JACC provider for Tivoli Access Manager is displayed.
4. Clear the **Enable embedded Tivoli Access Manager** option. If you want to ignore errors when unconfiguring, select the **Ignore errors during embedded Tivoli Access Manager disablement** option. Select this option only when the Tivoli Access Manager domain is in an irreparable state.
5. Click **OK**.
6. **Optional:** If you want security enabled without Tivoli Access Manager re-enable administrative securityy.
7. Restart all WebSphere Application Server instances for the changes to take effect.

Forcing the unconfiguration of the Tivoli Access Manager JACC provider:

If you find you cannot restart WebSphere Application Server after configuring the JACC provider for Tivoli Access Manager a utility is available to clear the security configuration and return WebSphere Application Server to an operable state.

The utility removes all of the PDLoginModuleWrapper entries as well as the Tivoli Access Manager authorization table from `security.xml` and `wsjaas.conf` files. This utility effectively removes the JACC provider for Tivoli Access Manager.

1. Back up the `security.xml` and `wsjaas.conf` files.
2. Enter the following command as one continuous line.

```
app_server_root/java/jre/bin/java
-classpath "app_server_root /$WAS_HOME/plug-in/com.ibm.ws.runtime_1.0.0.jar"
 com.tivoli.pd.as.jacc.cfg.CleanSecXML
 fully_qualified_path/security.xml fully_qualified_path/wsjaas.conf
```

Authorizing access to administrative roles

You can assign users and groups to administrative roles to identify users who can perform WebSphere Application Server administrative functions.

Administrative roles enable you to control access to WebSphere Application Server administrative functions. Refer to the descriptions of these roles in rsec_adminroles.dita.

Before you assign users to administrative roles, you must set up your user registry. For information on the supported registry types, see “Selecting a registry or repository” on page 1083.

The following steps are needed to assign users to administrative roles.

In the administrative console, click **Users and Groups**. Click either **Administrative User Roles** or **Administrative Group Roles**.

1. To add a user or a group, click **Add** on the Console users or Console groups panel.
2. To add a new administrator user, enter a user identity in the User field, highlight **Administrator**, and click **OK**. If there is no validation error, the specified user is displayed with the assigned security role.
3. To add a new administrative group, either enter a group name in the **Specify group** field or select EVERYONE or ALL AUTHENTICATED from the **Special subject** menu, highlight **Administrator**, and click **OK**. If no validation error occurs, the specified group or special subject is displayed with the assigned security role.
4. To remove a user or group assignment, click **Remove** on the Console Users or the Console Groups panel. On the Console Users or the Console Groups panel, select the check box of the user or group to remove and click **OK**.
5. To manage the set of users or groups to display, click **Show filter function** on the User Roles or Group Roles panel. In the **Search term(s)** box, type a value, then click **Go**. For example, user* displays only users with the user prefix.
6. After the modifications are complete, click **Save** to save the mappings.
7. Restart the application server for changes to take effect.

After you assign users to administrative roles, you must restart the server for the new roles to take effect. However, the administrative resources are not protected until you enable security.

Administrative user roles settings and CORBA naming service user settings

Use the Administrative User Roles page to give users specific authority to administer application servers through tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled. Use the Common Object Request Broker Architecture (CORBA) naming service users settings page to manage CORBA naming service users settings.

To view the Console Users administrative console page, complete either of the following steps:

- Click **Security > Secure administration, applications, and infrastructure > Administrative User Roles**.
- Click **Users and Groups > Administrative User Roles**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Note: When a third-party authorization such as Tivoli Access Manager or Service Access Facility (SAF) for z/OS is used, the information in this panel might not represent the data in the provider. Also, any changes to this panel might not be reflected in the provider automatically. Follow the provider's instructions to propagate any changes made here to the provider.

User (Administrative user roles):

Specifies users.

The users that are entered must exist in the configured active user registry.

Data type: String

User (CORBA naming service users):

Specifies CORBA naming service users.

The users that are entered must exist in the configured active user registry.

Data type: String

Role (Administrative user roles):

Specifies user roles.

The following administrative roles provide different degrees of authority that are needed to perform certain application server administrative functions:

Administrator
The administrator role has operator permissions, configurator permissions, and the permission that is required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Operator
The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Configurator
The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

Monitor
The monitor role has the least permissions. This role primarily confines the user to viewing the application server configuration and current state.

adminsecuritymanager
The adminsecuritymanager role has privileges for managing users and groups from within the administrative console and determines who has access to modify users and groups using administrative role mapping. Only the adminsecuritymanager role can map users and groups to administrative roles, and by default, AdminId is granted to the adminsecuritymanager.

iscadmins
The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

Data type: String
Range: Administrator, Operator, Configurator, Monitor, and iscadmins

Note: Other arbitrary administrative roles might also be visible in the administrative console collection table. Other contributors to the console might create these additional roles, which can be used for applications that are deployed to the console.

Role (CORBA naming service users):

Specifies naming service user roles.

A number of naming roles are defined to provide degrees of authority that are needed to perform certain application server naming service functions. The authorization policy is only enforced when global security is enabled. The following roles are valid: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete.

The roles now have authority levels from low to high:

CosNamingRead
You can query the application server name space by using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.

CosNamingWrite
You can perform write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations.

CosNamingCreate
You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations.

CosNamingDelete
You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations.

Data type: String
Range: CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete

Login status (Administrative user roles):

Species whether the user is active or inactive.

Administrative group roles and CORBA naming service groups

Use the Administrative Group Roles page to give groups specific authority to administer application servers through tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when administrative security is enabled. Use the Common Object Request Broker Architecture (CORBA) naming service groups page to manage CORBA Naming Service groups settings.

To view the Console Groups administrative console page, complete either of the following steps:

- Click **Security > Secure administration, applications, and infrastructure > Administrative Group Roles**.
- Click **Users and Groups > Administrative Group Roles**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Group (CORBA naming service groups):

Identifies CORBA naming service groups.

In previous releases of WebSphere Application Server, there were two default groups: ALL AUTHENTICATED and EVERYONE. However, EVERYONE is now the only default group, and it provides CosNamingRead privileges only.

Data type: String
Range: EVERYONE

Role (CORBA naming service groups):

Identifies naming service group roles.

A number of naming roles are defined to provide the degrees of authority that are needed to perform certain application server naming service functions. The authorization policy is only enforced when global security is enabled.

Four name space security roles are available: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The roles have authority levels from low to high:

Cos Naming Read
You can query the application server name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.

Cos Naming Write
You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Cos Naming Create
You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Cos Naming Delete
You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Data type: String
Range: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete

Group (Administrative group roles):

Specifies groups.

The ALL_AUTHENTICATED and the EVERYONE groups can have the following role privileges: Administrator, Configurator, Operator, and Monitor.

Data type: String
Range: ALL_AUTHENTICATED, EVERYONE

Role (Administrative group roles):

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain application server administrative functions:

Administrator
The administrator role has operator permissions, configurator permissions, and the permission that is required to access sensitive data, including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Operator
The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Configurator
The configurator role has monitor permissions and can change the application server configuration.

Monitor
The monitor role has the least permissions. This role primarily confines the user to viewing the application server configuration and current state.

iscadmins
The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

Data type:	String
Range:	Administrator, Operator, Configurator, Monitor, and iscadmins

Note: Other arbitrary administrative roles might also be visible in the administrative console collection table. Other contributors to the console might create these additional roles, which can be used for applications that are deployed to the console.

Assigning users to naming roles

Use this task to assign users to naming roles by using the administrative console.

The following steps are needed to assign users to naming roles. In the administrative console, click **Environment > Naming**, and click **CORBA Naming Service Users** or **CORBA Naming Service Groups**.

1. Click **Add** on the CORBA Naming Service Users or the CORBA Naming Service Groups panel.
2. To add a new naming service user, enter a user identity in the **User** field, highlight one or more naming roles, and click **OK**. If no validation errors occur, the specified user is displayed with the assigned security role.
3. To add a new naming service group, either select **Specify group** and enter a group name or select **Select from special subject** and then select **EVERYONE**. Click **OK**. If no validation errors occur, the specified group or special subject is displayed with the assigned security role.
4. To remove a user or group assignment, go to the **CORBA Naming Service Users** or **CORBA Naming Service Groups** panel. Select the check box next to the user or group that you want to remove and click **Remove**.
5. To manage the set of users or groups to display, expand the **Filter** folder on the right panel, and modify the filter text box. For example, setting the filter to `user*` displays only users with the `user` prefix.
6. After modifications are complete, click **Save** to save the mappings. Restart the server for the changes to take effect.

The default naming security policy is to grant all users read access to the CosNaming space and to grant any valid user the privilege to modify the contents of the CosNaming space. You can perform the previously mentioned steps to restrict user access to the CosNaming space. However, use caution when changing the naming security policy. Unless a Java 2 Platform, Enterprise Edition (J2EE) application has clearly specified its naming space access requirements, changing the default policy can result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime.

Propagating administrative role changes to Tivoli Access Manager

These steps provide an example of how to migrate the `admin-authz.xml` file.

Additions and changes to console users and groups are not automatically added to the Tivoli Access Manager object space after the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager is configured. Changes to console users and groups are saved in the `admin-authz.xml` file and this file must be migrated before any changes take effect. The JACC provider for Tivoli Access Manager includes the **migrateEAR** migration utility for incorporating console user and group changes into the Tivoli Access Manager object space.

Note: The **migrateEAR** utility is used to migrate the changes made to console users and groups after the JACC provider for Tivoli Access Manager is configured. The utility does not need to run for changes and additions to console user and groups made prior to the configuration of the JACC provider for Tivoli Access Manager because the changes made to the `admin-authz.xml` and `naming-authz.xml` files are automatically migrated at configuration time. Furthermore, the migration tool does not need to run before deploying standard Java 2 Platform, Enterprise Edition (J2EE) applications; J2EE application policy deployment is also performed automatically.

For example, if you wanted to migrate the `admin-authz.xml` file, perform the following steps:

1. Set up the environment.

   Before running the `migrateEAR` utility, set up the environment by running the `setupCmdLine.bat` or `setupCmdLine.sh` file that is located in the *app_server_root*`/bin` directory.

   Make sure that the *WAS_HOME* environment variable is set to the WebSphere Application Server installation directory.
2. Change to the *app_server_root*`/bin` directory where the `migrateEAR` utility is located.
3. Run the **migrateEAR** utility to migrate the data contained in the `admin-authz.xml` file. Use the parameter descriptions that are listed in "The migrateEAR utility for Tivoli Access Manager."

   A status message is displayed when the migration completes. Output of the utility is logged to the `pdwas_migrate.log` file, which is created in the directory where the utility is run. Check the log file after each migration. If the log file displays errors, check the last recorded transaction, correct the source of the error, and rerun the migration utility. If the migration is unsuccessful, verify that you supplied the correct values for the `-c` and `-j` options.
4. WebSphere Application Server does not require a restart for the changes to take effect.

The migrateEAR utility for Tivoli Access Manager

The **migrateEAR** utility migrates changes made to console users and groups in the `admin-authz.xml` and `naming-authz.xml` files into the Tivoli Access Manager object space.

Syntax

```
migrateEAR
-j fully_qualified_filename
-c pdPerm.properties_file_location
-a Tivoli_Access_Manager_administrator_ID
-p Tivoli_Access_Manager_administrator_password
-w WebSphere_Application_Server_administrator_user_name
-d user_registry_domain_suffix
[-r root_objectspace_name]
[-t ssl_timeout]
```

Parameters

-a *Tivoli_Access_Manager_administrator_ID*
The administrative user identifier. The administrative user must have the privileges required to create users, objects, and access control lists (ACLs). For example, `-a sec_master`.

This parameter is optional. When the parameter is not specified, you are prompted to supply it at run time.

-c *PdPerm.properties_file_location*
The Uniform Resource Indicator (URI) location of the `PdPerm.properties` file that is configured by the pdwascfg utility. When WebSphere Application Server is installed in the default location, the URI is:

Solaris Linux HP-UX

```
file:/opt/IBM/WebSphere/AppServer/java/jre/PdPerm.properties
```

AIX

```
file:/usr/IBM/WebSphere/AppServer/java/jre/PdPerm.properties
```

Windows

```
file:/"C:/Program Files/IBM/WebSphere/AppServer/java/jre/PdPerm.properties"
```

-d *user_registry_domain_suffix*
The domain suffix for the user registry to use. For example, for Lightweight Directory Access Protocol (LDAP) user registries, this value is the domain suffix, such as: `"o=ibm,c=us"`

Windows Windows platforms require that the domain suffix is enclosed within quotes.

You can use the **pdadmin user show** command to display the distinguished name (DN) for a user.

-j *fully_qualified_pathname*
The fully qualified path and file name of the Java 2 Platform, Enterprise Edition application archive file `,admin-authz.xml` or the roles definitions file `naming-authz.xml` that is used for a naming operation authorization. Optionally, this path can also be a directory of an expanded enterprise application. For example, when WebSphere Application Server is installed in the default location, the path to the data files to migrate includes:

Solaris Linux HP-UX

```
file:/opt/IBM/WebSphere/AppServer/profiles/profile_name/config/cells
/cell_name/admin-authz.xml
```

AIX

```
file:/usr/IBM/WebSphere/AppServer/profiles/profile_name/config/cells
/cell_name/admin-authz.xml
```

Windows

```
"C:/Program Files/IBM/WebSphere/AppServer/profiles/profile_name/config/cells
/cell_name/admin-authz.xml"
```

-p *Tivoli_Access_Manager_administrator_password*
The password for the Tivoli Access Manager administrative user. The administrative user must have the privileges that are required to create users, objects, and access control lists (ACLs). For example, you can specify the password for the `-a sec_master` administrative user as `-p myPassword`.

When this parameter is not specified, the user is prompted to supply the password for the administrative user name.

-r *root_objectspace_name*
The space name of the root object. The value is the name of the root of the protected object namespace hierarchy that is created for WebSphere Application Server policy data.

The default value for the root object space is `WebAppServer`.

Set the Tivoli Access Manager root object space name by modifying the `amwas.amjacc.template.properties` file prior to configuring the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager for the first time. Use this option if the default object space value is not used in the configuration of the Tivoli Access Manager JACC provider for Tivoli Access Manager.

Do not change the Tivoli Access Manager object space name after the Tivoli Access Manager JACC provider is configured.

-t *ssl_timeout*
The number of minutes for the Secure Sockets Layer (SSL) timeout. This parameter is used to disconnect and reconnect the SSL context between the Tivoli Access Manager authorization server and the policy server before the default connection times out.

The default is `60` minutes. The minimum value is `10` minutes. The maximum value cannot exceed the Tivoli Access Manager ssl-v3-timeout value. The default value for ssl-v3-timeout is `120` minutes.

If you are not familiar with the administration of this value, you can safely use the default value.

-w *WebSphere_Application_Server_administrator_user_name*
The user name that is configured in the WebSphere Application Server security user registry field as the administrator. This value matches the account that you created or imported in “Creating the security administrative user” on page 1370. Access permission for this user is needed to create or update the Tivoli Access Manager protected object space.

When the WebSphere Application Server administrative user does not already exist in the protected object space, it is created or imported. In this case, a random password is generated for the user and the account is set to `not valid`. Change this password to a known value and set the account to `valid`.

A protected object and access control list (ACL) are created. The administrative user is added to the pdwas-admin group with the following ACL attributes:

T Traverse permission

i Invoke permission

WebAppServer
You can overwrite the action group name. The default name is `WebAppServer`. This action group name and the matching root object space can be overwritten when the migration utility is run with the **-r** option.

Comments

This utility migrates security policy information from deployment descriptors or enterprise archive files to Tivoli Access Manager for WebSphere Application Server. The script calls com.tivoli.pdwas.migrate.Migrate the Java class.

Before invoking the script you must run the **setupCmdLine.bat** or the **setupCmdLine.sh** commands. These files can be found in the `%WAS_HOME%/bin` directory.

The script is dependent on finding the correct environment variables for the location of prerequisite software. The script calls Java code with the following options:

-Dpdwas.lang.home
The directory that contains the native language support libraries that are provided with the JACC

provider for Tivoli Access Manager. These libraries are located in a subdirectory under the JACC provider for Tivoli Access Manager installation directory. For example: `-Dpdwas.lang.home=%PDWAS_HOME%\java\nls`

-cp %CLASSPATH% com.tivoli.pdwas.migrate.Migrate
The CLASSPATH variable must be set correctly for your Java installation.

Windows Both the -j option and the -c option can reference the %WAS_HOME% variable to determine where WebSphere Application Server is installed. This information is used to:

- Build the full path name of the enterprise archive file.
- Build the full URI path name to the location of the `PdPerm.properties` file.

To enable a new user access to the administrative group in WebSphere Application Server, it is recommended that the user be added to the pdwas-admin group after JACC has been enabled. You can enter the administrative primary ID (adminID) in the group. This is required when the serverID is not the same as the adminID.

The following is an example of this command:

```
pdadmin> group modify pdwas-admin add adminID
```

Return codes

The utility can return the following exit status codes:

0 The command completed successfully.

1 The command failed.

Securing communications

WebSphere Application Server provides several methods to secure communication between a server and a client.

The following topics are covered in this section:

- Secure communications using Secure Sockets Layer
- Creating an SSL configuration
- Creating a keystore configuration
- Creating a self-signed certificate
- Creating a certificate authority request
- Extracting a signer certificate from a personal certificate
- Retrieving signers from a remote SSL port
- Adding a signer certificate to a keystore
- Exchanging signer certificates in a keystore
- Configuring certificate expiration monitoring
- Key management for cryptographic uses
- Creating a key set configuration
- Creating a key set group configuration

Secure communications using Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol provides transport layer security including authenticity, data signing, and data encryption to ensure a secure connection between a client and server that uses

WebSphere Application Server. The foundation technology for SSL is *public key cryptography*, which guarantees that when an entity encrypts data using its private key, only entities with the corresponding public key can decrypt that data.

WebSphere Application Server uses Java Secure Sockets Extension (JSSE) as the SSL implementation for secure connections. JSSE is part of the Java 2 Standard Edition (J2SE) specification and is included in the IBM implementation of the Java Runtime Extension (JRE). JSSE handles the handshake negotiation and protection capabilities that are provided by SSL to ensure secure connectivity exists across most protocols. JSSE relies on X.509 certificate-based asymmetric key pairs for secure connection protection and some data encryption. Key pairs effectively encrypt session-based secret keys that encrypt larger blocks of data. The SSL implementation manages the X.509 certificates.

Managing X.509 certificates

Secure communications for WebSphere Application Server require digitally-signed X.509 certificates. The contents of an X.509 certificate, such as its distinguished name and expiration, are either signed by a certificate authority (CA) or are self-signed. When a trusted CA signs an X.509 certificate, WebSphere Application Server identifies the certificate and freely distributes it. A certificate must be signed by a CA because the certificate represents the identity of an entity to the general public. Server-side ports that accept connections from the general public must use CA-signed certificates. Most clients or browsers already have the signer certificate that can validate the X.509 certificate so signer exchange is not necessary for a successful connection.

You can trust the identity of a self-signed X.509 certificate only when a peer in a controlled environment, such as internal network communications, accepts the signer certificate. To complete a trusted handshake, you must first send a copy of the entity certificate to every peer that connects to the entity. Self-signed certificates are less expensive than CA-signed certificates because they do not require signer exchange for a secure connection.

CA and self-signed X.509 certificates reside in Java *keystores*. JSSE provides a reference to the keystore in which a certificate resides. You can select from many types of keystores, including Java Cryptographic Extension (JCE)-based and hardware-based keystores. Typically, each JSSE configuration has two Java keystore references: a keystore and a *truststore*. The keystore reference represents a Java keystore object that holds personal certificates. The truststore reference represents a Java keystore object that holds signer certificates.

A personal certificate without a private key is an X.509 certificate that represents the entity that owns it during a handshake. Personal certificates contain both public and private keys. A signer certificate is an X.509 certificate that represents a peer entity or itself. Signer certificates contain just the public key and verify the signature of the identity that is received during a peer-to-peer handshake.

For more information, see “Extracting a signer certificate from a personal certificate” on page 1485

For more information about keystores, see csec_sslkeystoreconfs.dita.

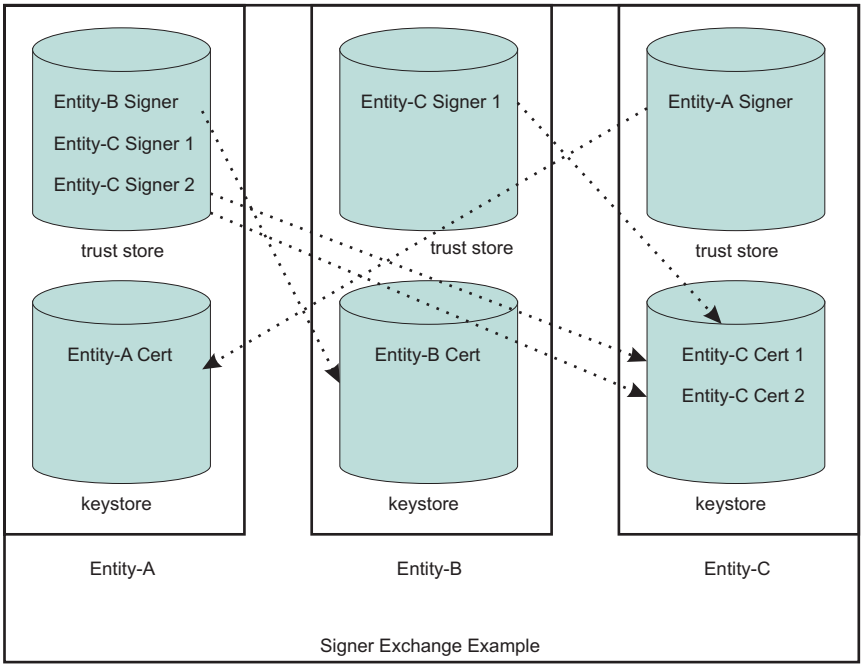
Signer exchange

When you configure an SSL connection, you can exchange signers to establish trust in a *personal certificate* for a specific entity. Signer exchange enables you to extract the X.509 certificate from the peer keystore and add it into the truststore of another entity so that the two peer entities can connect. The signer certificate also can originate from a CA as a root signer certificate or an intermediate signer certificate. You can also extract a *signer certificate* directly from a *self-signed certificate*, which is the X.509 certificate with the public key.

Figure 1 illustrates a hypothetical keystore and truststore configuration. An SSL configuration determines which entities can connect to other entities, and the peer connections that are trusted by an SSL

handshake. If you do not have the necessary signer certificate, the handshake fails because the peer cannot be trusted.

Figure 14. Signer exchange



In this example, the truststore for Entity A contains three signers. Entity A can connect to any peer as long as one of the three signers validates its personal certificate. For example, Entity A can connect to Entity B or Entity C because the signers can trust both signed personal certificates. The truststore for Entity-B contains one signer. Entity B is able to connect to Entity C only, and only when the peer endpoint is using certificate Entity-C Cert 1 as its identity. The ports that use the other personal certificate for Entity C are not trusted by Entity B. Entity C can connect to Entity A only.

In the example, the self-signed configuration seems to represent a one-to-one relationship between the signer and the certificate. However, when a CA signs a certificate, it typically signs many at a time. The advantage of using a single CA signer is that it can validate personal certificates that are generated by the CA for use by peers. However, if the signer is a public CA, you must be aware that the signed certificates might have been generated for another company other than your target entity. For your internal communications, private CAs and self-signed certificates are preferable to public CAs because they enable you to isolate the connections that you want to occur and prevent those that you do not want to occur.

SSL configurations

An SSL configuration comprises a set of configuration attributes that you can associate with an *endpoint* or set of endpoints in the WebSphere Application Server topology. The SSL configuration enables you to create an SSLContext object, which is the fundamental JSSE object that the server uses to obtain SSL socket factories. You can manage the following configuration attributes:

- An alias for the SSLContext object
- A handshake protocol version
- A keystore reference
- A truststore reference
- A key manager
- One or more trust managers

- A security level selection of a cipher suite grouping or a specific cipher suite list
- A certificate alias choice for client and server connections

To understand the specifics of each SSL configuration attribute, see “Secure Sockets Layer configurations” on page 1400.

Selecting SSL configurations

In previous releases of WebSphere Application Server, you can reference an SSL configuration only by selecting the SSL configuration alias directly. Each secure endpoint was denoted by an alias attribute that references a valid SSL configuration within a repertoire of SSL configurations. When you made a single configuration change, you had to re-configure many alias references across the various processes. Although the current release still supports direct selection, this approach is no longer recommended.

The current release provides improved capabilities for managing SSL configurations and more flexibility when you select SSL configurations. In this release, you can select from the following approaches:

Programmatic selection
: You can set an SSL configuration on the running thread prior to an outbound connection. WebSphere Application Server ensures that most system protocols, including Internet Inter-ORB Protocol (IIOP), Java Message Service (JMS), Hyper Text Transfer Protocol (HTTP), and Lightweight Directory Access Protocol (LDAP), accept the configuration. See “Example: Programmatically specifying an outbound SSL configuration using JSSEHelper API” on page 1448

Dynamic selection
: You can associate an SSL configuration dynamically with a specific target host, port, or outbound protocol by using a predefined selection criteria. When it establishes the connection, WebSphere Application Server checks to see if the target host and port match a predefined criteria that includes the domain portion of the host. Additionally, you can predefine the protocol for a specific outbound SSL configuration and certificate alias selection. See “Dynamic outbound selection of Secure Sockets Layer configurations” on page 1409 for more information.

Direct selection
: You can select an SSL configuration by using a specific alias, as in past releases. This method of selection is maintained for backwards compatibility because many applications and processes rely on alias references.

Scope selection
: You can associate an SSL configuration and its certificate alias, which is located in the keystore associated with that SSL configuration, with a WebSphere Application Server management scope. This approach is recommended to manage SSL configurations centrally. You can manage endpoints more efficiently because they are located in one topology view of the cell. The inheritance relationship between scopes reduces the number of SSL configuration assignments that you must set.

  Each time you associate an SSL configuration with a cell scope, the node scope within the cell automatically inherits the configuration properties. However, when you assign an SSL configuration to a node, the node configuration overrides the configuration that the node inherits from the cell. Similarly, all of the application servers for a node automatically inherit the SSL configuration for that node unless you override these assignments. Unless you override a specific configuration, the topology relies on the rules of inheritance from the cell level down to the endpoint level for each application server.

  The topology view displays an inbound tree and outbound tree. You can make different SSL configuration selections for each side of the SSL connection based on what that server connects to as an outbound connection and what the server connects to as an inbound connection. See “Central management of Secure Sockets Layer configurations” on page 1410 for more information.

The runtime uses an order of precedence for determining which SSL configuration to choose because you have many ways to select SSL configurations. Consider the following order of precedence when you select a configuration approach:

1. Programmatic selection. If an application sets an SSL configuration on the running thread using the com.ibm.websphere.ssl.JSSEHelper application programming interface (API), the SSL configuration is guaranteed the highest precedence.
2. Dynamic selection criteria for outbound host and port or protocol.
3. Direct selection.
4. Scope selection. Scope inheritance guarantees that the endpoint that you select is associated with an SSL configuration and is inherited by every scope beneath it that does not override this selection.

Default self-signed certificate configuration

By default, WebSphere Application Server creates a unique self-signed certificate for each node. WebSphere Application Server no longer relies on the default or dummy certificate that is shipped with the product. The `key.p12` default keystore and the trust.p12 truststore are stored in the configuration repository within the node directory.

All of the nodes put their signer certificates in this common truststore (`trust.p12`). Additionally, after you federate a node, the default SSL configuration is automatically modified to point to the common truststore, which is located in the cell directory. The node can now communicate with all other servers in the cell.

All default SSL configurations contain a keystore with the name suffix DefaultKeyStore and a truststore with the name suffix DefaultTrustStore. These default suffixes instruct the WebSphere Application Server runtime to add the signer of the personal certificate to the common truststore. If a keystore name does not end with DefaultKeyStore, the keystore signer certificates are not added to the common truststore when you federate the server. You can change the default SSL configuration, but you must ensure that the correct trust is established for administrative connections, among others.

For more information, see “Default self-signed certificate configuration” on page 1415 and “Web server plug-in default configuration” on page 1423.

Certificate expiration monitoring

Certificate monitoring ensures that the self-signed certificate for each node is not allowed to expire. The certificate expiration monitoring function issues a warning before certificates and signers are set to expire. Those certificates and signers that are located in keystores managed by the WebSphere Application Server configuration can be monitored. You can configure the expiration monitor to automatically replace a self-signed certificate with a new self-signed certificate that is based upon the same data that is used for the initial creation. The monitor also can automatically replace old signers with the signers from the new self-signed certificates in keystores that are managed by WebSphere Application Server. The existing signer exchanges that occurred by the runtime during federation and by administration are preserved. For more information, see “Certificate expiration monitoring” on page 1422.

WebSphere Application Server clients: signer-exchange requirements

A new self-signed certificate is generated for each node during its initial startup. To ensure trust, clients must be given these generated signers to establish a connection. Several enhancements in the current release make this process simpler. You can gain access to the signer certificates of various nodes to which the client must connect with any one of the following options (for more information, see “Secure installation for client signer retrieval” on page 1418):

- A signer exchange prompt enables you to import signer certificates that are not yet present in the truststores during a connection to a server. By default, this prompt is enabled for administrative connections and can be enabled for any client SSL configuration. When this prompt is enabled, any connection that is made to a server where the signer is not already present offers the signer of the

server along with the certificate information and a Secure Hash Algorithm (SHA) digest of the certificate for verification. The user is given a choice whether to accept these credentials. If the credentials are accepted, the signer is added to the truststore of the client until the signer is explicitly removed. The signer exchange prompt does not occur again when connecting to the same server unless the personal certificate changes.

Attention: It is unsafe to trust a signer exchange prompt without verifying the SHA digest. An unverified prompt can originate from a browser when a certificate is not trusted.

- You can run a retrieveSigners administrative script from a client prior to making connections to servers. To download signers, no administrative authority is required. To upload signers, you must have Administrator role authority. The script downloads all of the signers from a specified server truststore into the specified client truststore and can be called to download only a specific alias from a truststore. You can also call the script to upload signers to server truststores. When you select the CellDefaultTrustStore truststore as the specified server truststore and common truststore for a cell, all of the signers for that cell are downloaded to the specified client truststore, which is typically ClientDefaultTrustStore. For more information, see rxml_retrievesigners.dita.
- You can physically distribute to clients the trust.p12 common truststore that is located in the cell directory of the configuration repository. When doing this distribution, however, you must ensure that the correct password has been specified in the `ssl.client.props` client SSL configuration file. The default password for this truststore is `WebAS`. Change the default password prior to distribution. Physical distribution is not as effective as the previous options. When changes are made to the personal certificates on the server, automated exchange can fail.

Dynamic SSL configuration changes

The SSL runtime for WebSphere Application Server maintains listeners for most SSL connections. A change to the SSL configuration causes the inbound connection listeners to create a new SSLContext object. Existing connections continue to use the current SSLContext object. Outbound connections automatically use the new configuration properties when they are attempted.

Make dynamic changes to the SSL configuration during off-peak hours to reduce the possibility of timing-related problems and to prevent the possibility of the server starting again. If you enable the runtime to accept dynamic changes, then change the SSL configuration and save the `security.xml` file. Your changes take effect when the new `security.xml` file reaches each node.

Note: If configuration changes cause SSL handshake failures, administrative connectivity failures also can occur, which can lead to outages. In this case, you must re-configure the SSL connections then perform manual node synchronization to correct the problem. You must carefully complete any dynamic changes. It is highly recommended that you perform changes to SSL configurations on a test environment prior to making the same changes to a production system. For more information, see “Dynamic configuration updates” on page 1424.

Built-in certificate management

Certificate management that is comparable to iKeyMan functionality is now integrated into the keystore management panels of the administrative console. Use built-in certificate management to manage personal certificates, certificate requests, and signer certificates that are located in keystores. Additionally, you can remotely manage keystores. For example, you can manage a file-based keystore that is located outside the configuration repository on any node from the deployment manager. You also can remotely manage hardware cryptographic keystores from the deployment manager.

With built-in certificate management, you can replace a self-signed certificate along with all of the signer certificates scattered across many truststores and retrieve a signer from a remote port by connecting to the remote SSL host and port and intercepting the signer during the handshake. The certificate is first validated according to the certificate SHA digest, then the administrator must accept the validated certificate before it can be placed into a truststore.

When you make a certificate request, you can send it to a certificate authority (CA). When the certificate is returned, you can accept it within the administrative console. For more information, see “Certificate management” on page 1426.

Tip: Although iKeyMan functionality still ships with WebSphere Application Server, configure keystores from the administrative console using the built-in certificate management functionality. iKeyMan is still an option when it is not convenient to use the administrative console. For more information, see csec_sslikeymancertman.dita.

AdminTask configuration management

The SSL configuration management panels in the administrative console rely primarily on administrative tasks, which are maintained and enhanced to support the administrative console function. You can use wsadmin commands from a Java console prompt to automate the management of keystores, SSL configurations, certificates, and so on.

Secure Sockets Layer configurations

Secure Sockets Layer (SSL) configurations contain attributes that enable you to control the behavior of both client and server SSL endpoints. You can identify SSL configurations by their user-assigned names or aliases, and assign them to specific management scopes. The scope that an SSL configuration inherits depends upon whether you create it using a cell, node, server, or endpoint link in the configuration topology.

When you create an SSL configuration, you can set the following SSL connection attributes:

- Keystore
- Default client certificate for outbound connections
- Default server certificate for inbound connections
- Truststore
- Key manager for selecting a certificate
- Trust manager or managers for establishing trust during the handshake
- Handshaking protocol
- Ciphers for negotiating the handshake
- Client authentication support and requirements

You can manage an SSL configuration using any of the following methods:

- Central management selection
- Direct reference selection
- Dynamic outbound connection selection
- Programmatic selection

You can view an SSL configuration in the topology at the point where it was created and at all of the scopes below that point. If you want the entire cell to view an SSL configuration, you must create the configuration at the cell level in the topology. Using the administrative console, you can manage all of the SSL configurations for WebSphere Application Server. From the administrative console, click **Security > SSL certificates and key management > Manage endpoint security configurations > Inbound | Outbound >** *SSL_configuration*. Using the `ssl.client.props` properties file, you can manage client SSL configurations. The `ssl.client.props` file is located in the ${USER_INSTALL_ROOT}/properties directory for each profile. For more information about configuring this file, see the “ssl.client.props client configuration file” on page 1459.

SSL configuration in the security.xml file

In the security.xml file, you can define specific attributes to configure an SSL configuration repertoire entry at a specific management scope. The scope determines the point at which other levels in the cell topology can see the configuration, as shown in the following example:

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1" type="JSSE">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="false"
clientAuthenticationSupported="false" securityLevel="HIGH" enabledCiphers=""
jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS" keyStore="KeyStore_1"
trustStore="KeyStore_2" trustManager="TrustManager_1" keyManager="KeyManager_1"
clientKeyAlias="default" serverKeyAlias="default"/>
</repertoire>
```

The SSL configuration attributes that display in the previous code are described in Table 1.

Table 22. security.xml Attributes

security.xml attribute	Description	Default	Associated SSL property
xmi:id	The xml:id attribute represents the unique identifier for this XML entry and determines how the SSL configuration is linked to other XML objects, such as SSLConfigGroup. This system-defined value must be unique.	The administrative configuration service defines the default value.	None. This value is used only for XML associations.
alias	The alias attribute defines the name of the SSL configuration. Direct selection uses the alias attribute and the node is not prefixed to the alias. Rather, the management scope takes care of ensuring that the name is unique within the scope.		com.ibm.ssl.alias
managementScope	The managementScope attribute defines the management scope for the SSL configuration and determines the visibility of the SSL configuration at runtime.		The managementScope attribute is not mapped to an SSL property. However, it confirms whether or not the SSL configuration is associated with a process.
type	The type attribute defines the Java Secure Socket Extension (JSSE) or System Secure Sockets Layer (SSSL) configuration option. JSSE is the SSL configuration type for most secure communications within WebSphere Application Server.	The default is JSSE.	com.ibm.ssl.sslType
clientAuthentication	The clientAuthentication attribute determines whether SSL client authentication is required.	The default is false.	com.ibm.ssl.clientAuthentication

Table 22. security.xml Attributes (continued)

security.xml attribute	Description	Default	Associated SSL property
clientAuthenticationSupported	The clientAuthenticationSupported attribute determines whether SSL client authentication is supported. The client does not have to supply a client certificate if it does not have a client certificate. **Attention:** When you set the clientAuthentication attribute to `true`, you override the value that is set for the clientAuthenticationSupported attribute.	The default is false.	com.ibm.ssl.client.AuthenticationSupported
securityLevel	The securityLevel attribute determines the cipher suite group. Valid values include HIGH (128-bit ciphers), MEDIUM (40-bit ciphers), LOW (for all ciphers without encryption), and CUSTOM (if the cipher suite group is customized. When you set the enabledCiphers attribute with a specific list of ciphers, the system ignores this attribute.	The default is HIGH.	com.ibm.ssl.securityLevel
enabledCiphers	You can set the enabledCiphers attribute to specify a unique list of cipher suites. Separate each cipher suite in the list with a space.	The default is the securityLevel attribute for cipher suite selection.	com.ibm.ssl.enabledCipherSuites
jsseProvider	The jsseProvider attribute defines a specific JSSE provider.	The default is IBMJSSE2.	com.ibm.ssl.contextProvider
sslProtocol	The sslProtocol attribute defines the SSL handshake protocol. Valid options include: SSLv2 (client-side only), SSLv3, SSL, SSL_TLS, TLSv1, and TLS values. The SSL option includes SSLv2 and SSLv3 values. The TLS option includes the TLSv1 value. SSL_TLS, which is the most interoperable protocol, includes all these values and defaults to a Transport Layer Security (TLS) handshake.	The default is SSL_TLS.	com.ibm.ssl.protocol
keyStore	The keyStore attribute defines the keystore and attributes of the keyStore instance that the SSL configuration uses for key selection.		For more information, see Keystore configurations.
trustStore	The trustStore attribute defines the key store that the SSL configuration uses for certificate signing verification.		A trustStore is a logical JSSE term. It signifies a key store that contains signer certificates. Signer certificates validate certificates that are sent to WebSphere Application Server during an SSL handshake.

Table 22. security.xml Attributes (continued)

security.xml attribute	Description	Default	Associated SSL property
keyManager	The keyManager attribute defines the key manager that WebSphere Application Server uses to select keys from a key store. A JSSE key manager controls the javax.net.ssl.X509KeyManager interface. A custom key manager controls the javax.net.ssl.X509KeyManager and the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interfaces. The com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface provides more information from WebSphere Application Server.	The default is IbmX509.	com.ibm.ssl.keyManager defines a well-known key manager and accepts the algorithm and algorithm|provider formats, for example IbmX509 and IbmX509|IBMJSSE2. com.ibm.ssl.customKeyManager defines a custom key manager and takes precedence over the other keyManager properties. This class must implement javax.net.ssl.X509KeyManager and can implement com.ibm.wsspi.ssl.KeyManagerExtendedInfo. For more information, see csec_sslx509certIDkeyman.dita
trustManager	The trustManager determines which trust manager or list of trust managers to use for determining whether to trust the peer side of the connection. A JSSE trust manager implements the javax.net.ssl.X509TrustManager interface. A custom trust manager might also implement com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface to get more information from the WebSphere Application Server environment.	The default is IbmX509. You can specify the IbmPKIX trust manager for certificate revocation list (CRL) verification when the certificate contains a CRL distribution point.	com.ibm.ssl.trustManager defines a well-known trust manager, which is required for most handshake situations. com.ibm.ssl.trustManager performs certificate expiration checking and signature validation. You can define com.ibm.ssl.customTrustManagers with additional custom trust managers that are called during an SSL handshake. Separate additional trust managers with the vertical bar (|) character. For more information, see csec_sslx509certtrustdecisions.dita

Trust manager control of X.509 certificate trust decisions:

The role of the trust manager is to validate the Secure Sockets Layer (SSL) certificate that is sent by the peer, which includes verifying the signature and checking the expiration date of the certificate. A Java Secure Socket Extension (JSSE) trust manager determines if the remote peer can be trusted during an SSL handshake.

WebSphere Application Server has the ability to call multiple trust managers during an SSL connection. The default trust manager does the standard certificate validation; custom trust manager plug-ins run customized validation such as host name verification. For more information, see “Example: Developing a custom trust manager for custom SSL trust decisions” on page 1439

When a trust manager is configured in a server-side SSL configuration, the server calls the isClientTrusted method. When a trust manager is configured in a client-side SSL configuration, the client calls the isServerTrusted method. The peer certificate chain is passed to these methods. If the trust manager chooses not to trust the peer information, it might produce an exception to force a handshake failure.

Optionally, WebSphere Application Server provides the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface so that additional information can be passed to the trust manager. For more information, see the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface.

Default IbmX509 trust manager

The default IbmX509 trust manager, which is used in the following code sample, establishes trust by performing standard certificate validation.

```
<trustManagers xmi:id="TrustManager_1132357815717" name="IbmX509" provider="IBMJSSE2"
algorithm="IbmX509" managementScope="ManagementScope_1132357815717"/>
```

The trust manager provides a signer certificate to verify the peer certificate that is sent during the handshake. The signers who are added to the truststore for the SSL configuration must be trustworthy. If you do not trust the signers or do not want to allow others to connect to your servers, consider removing default root certificates from certificate authorities (CA). You might also remove any self-signed certificates if you cannot verify their origination.

Default IbmPKIX trust manager

You can use the default IbmPKIX trust manager to replace the IbmX509 trust manager, which is shown in the following code sample:

```
<trustManagers xmi:id="TrustManager_1132357815719" name="IbmPKIX" provider="IBMJSSE2"
algorithm="IbmPKIX" trustManagerClass="" managementScope="ManagementScope_1132357815717">
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1132357815717"
name="com.ibm.security.enableCRLDP" value="true" type="boolean"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1132357815718"
name="com.ibm.jsse2.checkRevocation" value="true" type="boolean"/>
  </trustManagers>
```

In addition to its role of standard certificate verification, the IbmPKIX trust manager checks for certificates that contain certificate revocation list (CRL) distribution points. This process is known as *extended CRL checking*. When you select a trust manager, its associated properties are automatically set as Java System properties so that the IBMCertPath and IBMJSSE2 providers are aware that CRL checking is enabled.

Custom trust manager

You can define a custom trust manager to perform additional trust checking, which is based upon the needs of the environment. For example, in one environment, you might enable connections from the same Transmission Control Protocol (TCP) subnet only. The com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface provides extended information about the connection that is not provided by the standard Java Secure Sockets Extension (JSSE) javax.net.ssl.X509TrustManager interface. The configured trustManagerClass attribute determines which class is instantiated by the runtime, as shown in the following code sample:

```
<trustManagers xmi:id="TrustManager_1132357815718" name="CustomTrustManager"
trustManagerClass="com.ibm.ws.ssl.core.CustomTrustManager"
managementScope="ManagementScope_1132357815717"/>
```

The trustManagerClass attribute must implement the javax.net.ssl.X509TrustManager interface and, optionally, can implement the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface.

Disabling the default trust manager

In some cases, you might not want to perform the standard certificate verification that is provided by the IbmX509 and IbmPKIX default trust managers. For example, you might be working with an internal automated test infrastructure that is not concerned with SSL client or server authentication, integrity, or confidentiality. The following sample code shows a basic custom trust manager such as com.ibm.ws.ssl.core.CustomTrustManager whose property is set to `true`.

```
com.ibm.ssl.skipDefaultTrustManagerWhenCustomDefined=true
```

You can set this property in the global properties at the top of the `ssl.client.props` file for clients or in the `security.xml` custom properties file for servers. You must configure a custom trust manager when you disable the default trust manager to prevent the server from calling the default trust manager even though it is configured. Disabling the default trust manager is not a common practice. Be sure to test the system with the disabled default trust manager in a test environment first. For more information on setting up a

custom trust manager, see “Creating a custom trust manager configuration” on page 1436

Key manager control of X.509 certificate identities:

The role of a Java Secure Socket Extension (JSSE) key manager is to retrieve the certificate that is used to identify the client or server during a Secure Sockets Layer (SSL) handshake.

WebSphere Application Server provides a default key manager that can select a certificate from a keystore when you define the following SSL configuration properties:

com.ibm.ssl.keyStoreClientAlias
Defines the alias that is chosen from the keystore for the client side of a connection. This alias must be present in the keystore.

com.ibm.ssl.keyStoreServerAlias
Defines the alias that is chosen from the keystore for the server side of a connection. This alias must be present in the keystore.

These two properties are set automatically when you use the administrative console because the default key manager is already configured.

With WebSphere Application Server, you can configure only one key manager at a time for a given SSL configuration. If you want custom certificate selection logic on the client side, you must write a new custom key manager. The custom key manager could provide function that prompts the user to choose a certificate dynamically. Also, you can implement an extended interface so that a key manager can provide information during connection time. For more information on the extended interface, see the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface. For more information on custom key manager development, see rsec_ssldevcustomkeymgr.dita.

Default IbmX509 key manager

The default IbmX509 key manager chooses a certificate to serve as the identity for an SSL handshake. The key manager is called to enable client authentication on either side of the SSL handshake; frequently on the server-side, and less frequently on the client side according to client and server requirements. If a keystore is not configured on the client-side and SSL client authentication is enabled, the key manager cannot select a certificate to send to the server. Therefore, the handshake fails.

The following sample code shows the key manager configuration in the `security.xml` file for an IbmX509 key manager.

```
<keyManagers xmi:id="KeyManager_1" name="IbmX509"
provider="IBMJSSE2" algorithm="IbmX509" keyManagerClass=""
managementScope="ManagementScope_1"/>
```

You do not specify the keyManagerClass class because the key manager is provided by the IBMJSSE2 provider. However, you can specify whether the key manager is a custom class implementation, in which case you must specify the keyManager class, or an algorithm name that WebSphere Application Server can start from the Java security provider framework.

Custom key manager

The following sample code shows the key manger configuration in the `security.xml` file for a custom class.

```
<keyManagers xmi:id="KeyManager_2" name="CustomKeyManager"
keyManagerClass="com.ibm.ws.ssl.core.CustomKeyManager"
managementScope="ManagementScope_1"/>
```

The custom class must implement the javax.net.ssl.X509KeyManager interface and, optionally, implement the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface to retrieve additional WebSphere Application Server information. This interface replaces the function of the default key manager because you can configure only one key manager at a time. Therefore, the custom key manager has sole responsibility for selecting the alias to use from the configured keystore. The benefit of a custom key manager is its ability, on the client side, to prompt for an alias. This process enables the user to decide which certificate to use in situations where the user knows the client certificate identity. For more information, see tsec_sslcreatecuskeymgr.dita.

Keystore configurations

Use keystore configurations to define how the runtime for WebSphere Application Server loads and manages keystore types for Secure Sockets Layer (SSL) configurations.

By default, the java.security.Security.getAlgorithms(″KeyStore″) attribute does not display a predefined list of keystore types in the administrative console. Instead, WebSphere Application Server retrieves all of the KeyStore types that can be referenced by the java.security.KeyStore object, including hardware cryptographic, z/OS platform, i5/OS platform, IBM Java Cryptography Extension (IBMJCE), and Java-based content management system (CMS)-provider keystores. If you specify a keystore provider in the `java.security` file or add it to the provider list programmatically, WebSphere Application Sever also retrieves custom keystores. The retrieval list depends upon the java.security configuration for that platform and process.

IBMJCE file-based keystores (JCEKS, JKS, and PKCS12)

A typical IBMJCE file-based keystore configuration is shown in the following sample code:

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}349dkckdd=" provider="IBMJCE"
location="${USER_INSTALL_ROOT}/config/cells/myhostNode01Cell
/nodes/myhostNode01/key.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" readOnly="false"
managementScope="ManagementScope_1"/>
```

For more information about default keystore configurations, see csec_ssldefselfsigncertconf.dita.

Table 1 describes the attributes that are used in the sample code.

Table 23. keystore configurations

Attribute name	Default	Description
xmi:id	Varies	A value that issued to reference the keystore from another area in the configuration, for example, from an SSL configuration. Make this value unique within the `security.xml` file.
name	For Java Secure Socket Extension (JSSE) keystore: NodeDefaultKeyStore. For JSSE truststore: NodeDefaultTrustStore.	A name that is used to identify the keystore by sight. The name can determine if the keystore is a default keystore based upon whether the name ends with DefaultKeyStore or DefaultTrustStore.
password	The default keystore password is `WebAS`. It is recommended that this be changed as soon as possible. See “Updating default key store passwords using scripting” on page 1527 for more information.	The password that is used to access the keystore name is also the default that is used to store keys within the keystore.
provider	The default provider is IBMJCE.	The Java provider that implements the type attribute (for example, PKCS12 type). The provider can be left unspecified and the first provider that implements the keystore type specified is used.

Table 23. keystore configurations (continued)

Attribute name	Default	Description
location	The default varies, but typically references a `key.p12` file or a `trust.p12` file in the node or cell directories of the configuration repository. These files are PKCS12 type keystores.	The keystore location reference. If the keystore is file-based, the location can reference any path in the file system of the node where the keystore is located. However, if the location is outside of the configuration repository, and you want to manage the keystore remotely from the administrative console or from the **wsadmi**n utility, then specify the hostList attribute that contains the host name of the node where it resides.
type	The default Java crypto device keystore type is `PKCS12`.	This type specifies the keystore. Valid types can be those returned by the java.security.Security.getAlgorithms(″KeyStore″) attribute. These types include the following keystore types, and availability depends on the process and platform `java.security` configuration: • JKS • JCEKS • PKCS12 • PKCS11 (Java crypto device) • CMSKS • IBMi5OSKeyStore • JCERACFKS • JCE4758KS (z/OS crypto device)
fileBased	The default is `true`.	This option is required for default keystores. It indicates a file-system keystore so you can use a FileInputStream or FileOutputStream for loading and storing the keystore.
hostList	The hostList attribute is used to specify a remote hostname so that the keystore can be remotely managed. There are no remotely managed keystores by default. All default keystores are managed locally in the configuration repository and synchronized out to each of the nodes.	The option manages a keystore remotely. You can set the host name of a valid node for a keystore. When you use either the administrative console or the wsadmin utility to manage certificates for this keystore, an MBean call is made to the node where the keystore exists for the approved operation. You can specify multiple hosts, although synchronization of the keystore operations are not guaranteed. For example, one of the hosts that is listed might be down when a specific operation is performed. Therefore, use multiple hosts in this list.
initializeAtStartup	The default is `true`.	This option informs the runtime to initialize the keystore during startup. This option can be important for hardware cryptographic device acceleration.
readOnly	The default is `false`.	This option informs the configuration that you cannot write to this keystore. That is, certain update operations on the keystore cannot be attempted and are not allowed. An example of a read-only keystore type is JCERACFKS on the z/OS platform. This type is read-only from the WebSphere certificate management standpoint, but you can also update it using the keystore management facility for RACF.

Table 23. keystore configurations (continued)

Attribute name	Default	Description
managementScope	The default scope is the node scope for a base Application Server environment and the cell scope for a Network Deployment environment.	This option references a particular management scope in which you can see this keystore. For example, if a hardware cryptographic device is physically located on a specific node, then create the keystore from a link to that node in the topology view under **Security > Security Communications > SSL configurations**. You can also use management scope to isolate a keystore reference. In some cases, you might need to allow only a specific application server to reference the keystore; the management scope is for that specific server.

CMS keystores

You can set some provider-specific attributes in CMS keystores.

If the CMSKS provider supports the createStashFileForCMS attribute, and you set the attribute to `true` for CMSKS keystores, WebSphere Application Server creates an `.sth` file in the keystore location that is referenced by the attribute. The `.sth` extension is appended to the keystore name. For example, if the CMSKS keystore is available for a plug-in configuration and you set createStashFileForCMS to `true`, the stash file that is represented in the following sample code is created in the `${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.sth` path.

```
<keyStores xmi:id="KeyStore_1132071489571" name="CMSKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMCMSProvider"
location="${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01
/nodes/myhostNode01/servers/webserver1/plugin-key.kdb" type="CMSKS"
fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132071489569"/>
```

When you create a CMS keystore, the CMS provider is `IBMi5OSJSSEProvider`, and the CMS type is `IBMi5OSKeyStore`, as shown in the following sample code:

```
<keyStores xmi:id="KeyStore_1132071489571" name="CMSKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMi5OSJSSEProvider"
location="${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01
/nodes/myhostNode01/servers/webserver1/plugin-key.kdb" type="IBMi5OSKeyStore"
fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132071489569"/>
```

Hardware cryptographic keystores

For cryptographic device configuration, see “Key management for cryptographic uses” on page 1502 and “Configuring a hardware cryptographic keystore” on page 1467

You can add a slot either as the custom property, com.ibm.ssl.keyStoreSlot, or as the configuration attribute, `slot="0"`. The custom property is read before the attribute for backwards compatibility.

In certain environments, you can use the hardware cryptographic card for hardware acceleration. Either set the useForAcceleration attribute to `true` or set the com.ibm.ssl.keyStoreUseForAcceleration custom property. When you set the attribute to `true`, you are not required to configure a password. However, you cannot use the device to store keys. For more information on configuring a hardware cryptographic keystore, see Configuring hardware cryptographic keystore configurations.

Default key store passwords:

When WebSphere Application Server starts for the first time as a standalone application server or as a network deployment, each server creates a key store and trust store for the default SSL configuration. When WebSphere Application Server creates these files, they use a default password (WebAS). Because the default password is well known, it is important that you change the password to protect the security of the key store files and the SSL configuration.

You can change the keystore passwords of:

- A single keystore or multiple key stores using a command task
- Multiple keystores using a command task
- A single keystore, using the administrative console

Dynamic outbound selection of Secure Sockets Layer configurations

WebSphere Application Server provides dynamic outbound selection that enables you to choose a specific Secure Sockets Layer (SSL) configuration and certificate alias for each outbound protocol, target host, target port, or any combination of these attributes. You can specify the dynamic selection information for outbound connections from a pure client or from a server that is acting as a client.

Before the SSL runtime for WebSphere Application Server starts an outbound connection, the runtime attempts to match the outbound protocol, target host, and target port attributes with the dynamic outbound selection information that is associated with an SSL configuration and certificate alias in the configuration.

The runtime caches both selection misses and selection hits, so the impact on performance can be minimal. However, a relationship exists between the amount of dynamic outbound selection information and its impact on the initial connection performance.

Target information during outbound connections

The dynamic outbound selection configurations are only effective when the outbound protocol uses the JSSEHelper application programming interface (API) when you select an SSL configuration with a specified connectionInfo hash map. This hash map must contain the following properties:

com.ibm.ssl.direction
The value for outbound connections is OUTBOUND.

com.ibm.ssl.remoteHost
The format should match what the protocol provides. Typically this is the canonical Domain Name Space (DNS), but it also could be the IP address.

com.ibm.ssl.remotePort
The port is target port.

com.ibm.ssl.endPointName
The value for an outbound connection must be one of the following protocol strings:

- IIOP
- HTTP
- SIP
- LDAP
- ADMIN_SOAP
- BUS_TO_BUS
- BUS_CLIENT
- BUS_TO_WEBSPHERE_MQ

Central management of Secure Sockets Layer configurations

By default, Secure Sockets Layer (SSL) configurations for servers are managed from a central location in the topology view in the administrative console. You can associate an SSL configuration and certificate alias with a management scope. This method is the most efficient method to manipulate and modify configurations when the server topology changes.

In prior releases, SSL configurations are managed in the `server.xml` file for each process. You have to edit individual `server.xml` documents to modify individual SSL configuration aliases in the configuration topology. In this release of WebSphere Application Server, management control of SSL configurations offers more flexibility and options. You can make coarse-grained changes using the cell-scope and fine-grained changes using a particular endpoint name, as defined in the `serverindex.xml` file for a specific application server process.

Because SSL configuration associations manifest inheritance behaviors, you can simplify the number of associations by referencing only the highest level management scope that needs a unique configuration. Obviously, the security environment influences issues such as SSL configuration uniqueness, and SSL configuration and certificate alias placement in the topology.

To configure the inbound and outbound topologies, which must be done separately in the administrative console, click **Security > SSL certificates and key management > Manage endpoint security configurations > Inbound | Outbound**.

The topology view provides the scoping mechanism. The SSL configuration inherits its visibility, which is its display in the topology, at the scope where you created the configuration and at all the scopes beneath this parent scope. When you create an SSL configuration at a specific node, the configuration can be seen by that node agent and by every application server that is part of that node. Any application server or node that is not part of this particular node cannot see this SSL configuration. You can configure different certificate aliases and SSL configurations for inbound versus outbound connections.

Default centrally-managed SSL configuration

The default management scope is the node scope. When a node is federated into a cell, the default SSL configurations for the node are maintained, as shown in the following sample code for the sslConfigGroups and management scopes attributes:

```
<sslConfigGroups xmi:id="SSLConfigGroup_1" name="myhostNode01"
direction="inbound" certificateAlias="default" sslConfig="SSLConfig_1"
managementScope="ManagementScope_1"/>
<sslConfigGroups xmi:id="SSLConfigGroup_2" name="myhostNode01"
direction="outbound" certificateAlias="default" sslConfig="SSLConfig_1"
managementScope="ManagementScope_1"/>

<managementScopes xmi:id="ManagementScope_1"
scopeName="(cell):myhostNode01Cell:(node):myhostNode01" scopeType="node"/>
```

The SSL configuration xmi:id ″SSLConfig_1″ is also federated and applicable:

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="true"
securityLevel="HIGH" enabledCiphers="" jsseProvider="IBMJSSE2"
sslProtocol="SSL_TLS" keyStore="KeyStore_1" trustStore="KeyStore_2"
trustManager="TrustManager_1" keyManager="KeyManager_1"/>
</repertoire>
```

The keystores that are associated with the SSLConfig_1 SSL configuration are also federated, and `key.p12` is located in the node directory of the configuration repository:

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMJCE"
location="${USER_INSTALL_ROOT}/config/cells/myhostNode01Cell/nodes
```

```
/myhostNode01/key.p12" type="PKCS12" fileBased="true" hostList=""
initializeAtStartup="true" managementScope="ManagementScope_1"/>
<keyStores xmi:id="KeyStore_2" name="NodeDefaultTrustStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMJCE"
location="${USER_INSTALL_ROOT}/config/cells/myhostNode01Cell
/nodes/myhostNode01/trust.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

Secure Sockets Layer node, application server, and cluster isolation

Secure Sockets Layer (SSL) enables you to ensure that any client that attempts to connect to a server during the handshake first performs server authentication. You can isolate communications between servers that must not communicate with each other over secure ports using SSL configurations at the node, application server, and cluster scopes.

Before you attempt to isolate communications controlled by WebSphere Application Server, you must have a good understanding of the deployment topology and application environment. To isolate a node, application server, or cluster, you must be able to control the signers that are contained in the trust stores that are associated with the SSL configuration. When the client does not contain the server signer, it cannot establish a connection to the server. If you use self-signed certificates, the server that created the personal certificate controls the signer, although you do have to manage the self-signed certificates. If you obtain certificates from a certificate authority (CA), you must obtain multiple CA signers because all of the servers can connect to each other if they share the same signer.

Authenticating only the server-side of a connection is not adequate protection when you need to isolate a server. Any client can obtain a signer certificate for the server and add it to its trust store. SSL client authentication must also be enabled between servers so that the server can control its connections by deciding which client certificates it can trust. For more information, see tsec_sslclientauthinbound.dita, which applies as well to enabling SSL client authentication at the cell level.

Isolation also requires that you use centrally managed SSL configurations for all or most endpoints in the cell. Centrally managed configurations can be scoped, unlike direct or end point configuration selection, and they enable you to create SSL configurations, key stores, and trust stores at a particular scope. Because of the inheritance hierarchy of WebSphere Application Server cells, if you select only the properties that you need for an SSL configuration, only these properties are defined at your selected scope or lower. For example, if you configure at the node scope, your configuration applies to the application server and individual end point scopes below the node scope. For more information, see tsec_sslassocconfigscope.dita, tsec_sslselconfigdirect.dita, and tsec_sslassocconfigout.dita

When you configure the key stores, which contain cryptographic keys, you must work at the same scope at which you define the SSL configuration and not at a higher scope. For example, if you create a key store that contains a certificate whose host name is part of the distinguished name (DN), then store that keystore in the node directory of the configuration repository. If you decide to create a certificate for the application server, then store that keystore on the application server in the application server directory.

When you configure the trust stores, which control trust decisions on the server, you must consider how much you want to isolate the application servers. You cannot isolate the application servers from the node agents or the deployment manager. However, you can configure the SOAP connector end points with the same personal certificate or to share trust. Naming persistence requires IIOP connections when they pass through the deployment manager. Because application servers always connect to the node agents when the server starts, the IIOP protocol requires that WebSphere Application Server establish trust between the application servers and the node agents.

Establishing node SSL isolation

By default, WebSphere Application Server uses a single self-signed certificate for each node so you can isolate nodes easily. A common trust store, which is located in the cell directory of the configuration

repository, contains all of the signers for each node that is federated into the cell. After federation, each cell process trusts all of the other cell processes because every SSL configuration references the common trust store.

You can modify the default configuration so that each node has its own trust store, and every application server on the node trusts only the node agent that uses the same personal certificate. You must also add the signer to the node trust store so that WebSphere Application Server can establish trust with the deployment manager. To isolate the node, ensure that the following conditions are met:

- The deployment manager must initiate connections to any process
- The node agent must initiate connections to the deployment manager and its own application servers
- The application servers must initiate connections to the applications servers on the same node, to its own node agent, and the deployment manager

Figure 1 shows Node Agent A contains a key.p12 keystore and a trust.p12 trust store at the node level of the configuration repository for node A.

Figure 15.

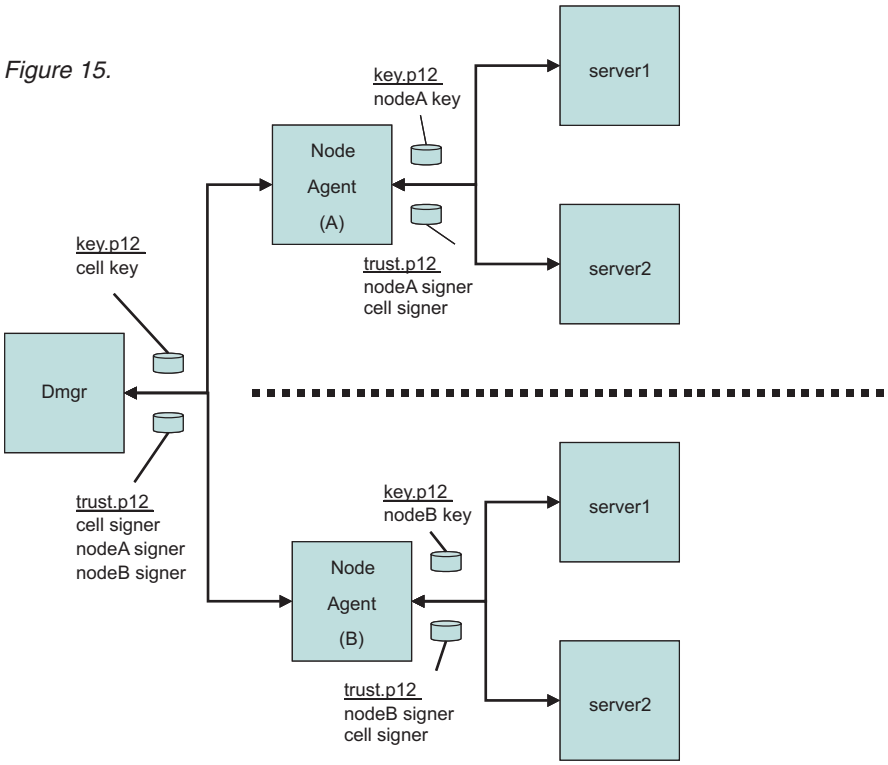


Figure 1: SSL Node Isolation

When you associate an SSL configuration with this keystore and truststore, you break the link with the cell-scoped trust store. To isolate the node completely, repeat this process for each node in the cell. WebSphere Application Server SSL configurations override the cell scope and use the node scope instead so that each process at this scope uses the SSL configuration and certificate alias that you selected at this scope. You establish proper administrative trust by ensuring that nodeA signer is in the common trust store and the cell signer is in the nodeA trust store. The same logic applies to node B as well. For more information, see tsec_sslassocconfigscope.dita.

Establishing application server SSL isolation

Isolating application server processes from one another is more challenging than isolating nodes. You must consider the following application design and topology conditions:

- An application server process might need to communicate with the node agent and deployment manager
- Isolating application server processes from each other might disable single sign-on capabilities for horizontal propagation

If you configure outbound SSL configurations dynamically, you can accommodate these conditions. When you define a specific outbound protocol, target host, and port for each different SSL configuration, you can override the scoped configuration.

Figure 2 shows how you might isolate an application server completely, although in practice this approach would be more complicated.

Figure 16.

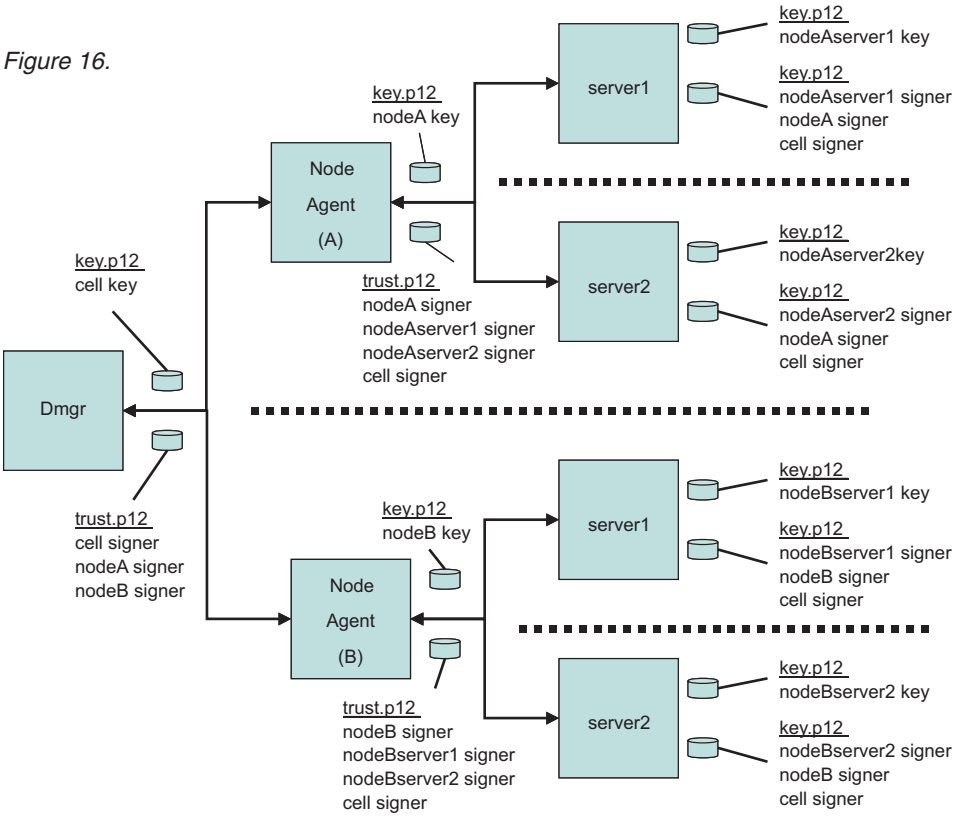


Figure 2: SSL Application Server Isolation

The dynamic configuration enables server1 on Node A to communicate with server 1 on Node B only over IIOP. The dynamic outbound rule is IIOP,nodeBhostname,*. For more information, see tsec_sslassocconfigout.dita

Establishing cluster SSL isolation

You can configure application servers into clusters instead of scoping them centrally at the node or dynamically at the server to establish cluster SSL isolation. While clustered servers can communicate with each other, application servers outside of the cluster cannot communicate with the cluster, thus isolating the clustered servers. For example, you might need to separate applications from different departments

while maintaining a basic level of trust among the clustered servers. Using the dynamic outbound SSL configuration method described for servers above, you can easily extend the isolated cluster as needed.

Figure 3 shows a sample cluster configuration where cluster 1 contains a key.p12 with its own self-signed certificate, and a trust.p12 that is located in the config/cells/<cellname>/clusters/<clustername> directory.

Figure 17.

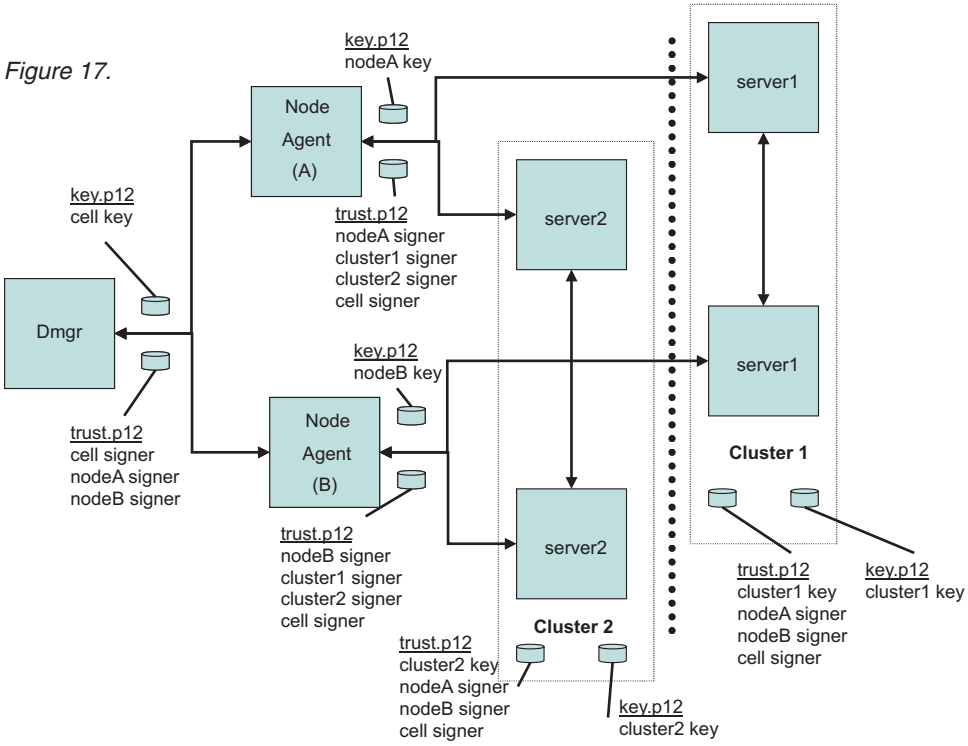


Figure 3: SSL Cluster Isolation

In the example, cluster1 might contain web applications, and cluster2 might contain EJB applications. Considering the various protocols, you decide to enable IIOP traffic between the two clusters. Your task is to define a dynamic outbound SSL configuration at the cluster1 scope with the following properties:

```
IIOP,nodeAhostname,9403|IIOP,nodeAhostname,9404|IIOP,nodeBhostname,9403|IIOP,nodeBhostname,9404
```

You must create another SSL configuration at the cluster1 scope that contains a new trust.p12 file with the cluster2 signer. Consequently, outbound IIOP requests go either to nodeAhostname ports 9403 and 9404 or to nodeBhostname ports 9403 and 9404. The IIOP SSL port numbers on these two application server processes in cluster2 identify the ports.

As you review Figure 3, notice the following features of the cluster isolation configuration:

- The trust.p12 for cluster1 contains signers that allow communications with itself (cluster1 signer), between both node agents (nodeAsigner and nodeBsigner), and with the deployment manager (cell signer).
- The trust.p12 for cluster2 contains signers that allow communications with itself (cluster2 signer), between both node agents (nodeAsigner and nodeBsigner), and with the deployment manager (cell signer).
- Node agent A and Node agent B can communicate with themselves, the deployment manager, and both clusters.

For more information, see tsec_sslassocconfigout.dita.

Although this article presents an overview of isolation methods from an SSL perspective, you must also ensure that non-SSL ports are closed or applications require the confidentiality constraint in the deployment descriptor. For example, you can set the CSIv2 inbound transport panel to require SSL and disable the channel ports that are not secure from the server ports configuration.

Also, you must enable SSL client authentication for SSL to enforce the isolation requirements on both sides of a connection. Without mutual SSL client authentication, a client can easily obtain a signer for the server programmatically and thus bypass the goal of isolation. With SSL client authentication, the server would require the client's signer for the connection to succeed. For HTTP/S protocol, the client is typically a browser, a Web Service, or a URL connection. For the IIOP/S protocol, the client is typically another application server or a Java client. WebSphere Application Server must know the clients to determine if SSL client authentication enablement is possible. Any applications that are available through a public protocol must not enable SSL client authentication because the client may fail to obtain a certificate to authenticate to the server.

Note: It is beyond the scope of this article to describe all of the factors you must consider to achieve complete isolation.

Default self-signed certificate configuration

When a WebSphere Application Server process starts for the first time, the Secure Sockets Layer (SSL) runtime initializes the default keystores and truststores that are specified in the SSL configuration.

Default keystore and truststore properties

WebSphere Application Server creates the key.p12 default keystore file and the trust.p12 default truststore file during profile creation. A default, self-signed certificate is also created in the key.p12 file at this time. The signer or public key is extracted from the key.p12 file and added to the trust.p12 file. If the files do not exist during process startup, they are recreated during startup.

You can easily identify keystore and truststore defaults because of their suffixes: DefaultKeyStore and DefaultTrustStore. Also, in the SSL configuration, you must set the fileBased attribute to `true` so that the runtime uses the default keystores and truststores only.

On a base application server, default key and truststores are stored in the node directory of the configuration repository. For example, the default `key.p12` and `trust.p12` stores are created with the AppSrv01 profile name, the myhostNode01Cell name, and the myhostNode01 node name. The key and truststores are located in the following directories:

```
C:\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell
\nodes\myhostNode01\key.p12
C:\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell
\nodes\myhostNode01\trust.p12
```

The default password is `WebAS` for all default keystores generated by WebSphere Application Server. Change the default password after the initial configuration for a more secure environment.

Default self-signed certificate

The default self-signed certificate is created during profile creation for both the server and client for that profile.

You can recreate the certificates with different information simply by deleting the `*.p12` files in /config and /etc. Change the four properties below to the values you want the certificates to contain, then restart the processes. This causes the server certificate in `/config` and the client certificate in `/etc` to differ.

If you want to set up SSL client authentication between the client and server, you must perform a signer exchange. The certificate properties below that exist in the `ssl.client.props` file do not exist in the server

configuration. However, you can use these values in the server configuration by adding them as custom security properties in the administrative console. The certificate properties appear in the `ssl.client.props` file, but do not appear in the server configuration. However, you can modify these values in the server configuration by adding them as custom properties in the administrative console.

Click **Security > Secure administration, applications, and infrastructure > Custom properties** to change the following properties:

```
com.ibm.ssl.defaultCertReqAlias=default_alias
com.ibm.ssl.defaultCertReqSubjectDN=cn=${hostname},o=IBM,c=US
com.ibm.ssl.defaultCertReqDays=365
com.ibm.ssl.defaultCertReqKeySize=1024
```

If a default_alias value already exists, the runtime appends _#, where the number sign (#) is a number that increments until it is unique in the keystore. ${hostname} is a variable that is resolved to the host name where it was originally created. The default expiration date of self-signed certificates is one year from their creation date.

The runtime monitors the expiration dates of self-signed certificates using the Certificate Expiration Monitor. These self-signed certificates are automatically replaced along with the signer certificates when they are within the expiration threshold, which is typically 30 days before expiration. You can increase the default key size beyond 1024 bits only when the Java Runtime Environment policy files are unrestricted, that is, not exported. For more information, see csec_sslcertmonitoring.dita.

Default keystore and truststore configurations for new Base Application Server processes

The following sample code shows the default SSL configuration for a base application server. References to the default keystores and truststores files are highlighted.

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="false"
securityLevel="HIGH" enabledCiphers="" jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS"
keyStore="KeyStore_1" trustStore="KeyStore_2" trustManager="TrustManager_1"
keyManager="KeyManager_1"/>
</repertoire>
```

Default keystore

In the following sample code, the keystore object that represents the default keystore is similar to the XML object.

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}349dkckdd=" provider="IBMJCE" location="${USER_INSTALL_ROOT}/config
/cells/myhostNode01Cell/nodes/myhostNode01/key.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

The NodeDefaultKeyStore keystore contains the personal certificate that represents the identity of the secure endpoint. Any keystore reference can use the ${USER_INSTALL_ROOT} variable, which is expanded by the runtime. The `PKCS12` default keystore type is in the most interoperable format, which means that it can be imported into most browsers. The myhostNode01Cell password is encoded.

The management scope determines which server runtime loads the keystore configuration into memory, as shown in the following code sample:

```
<managementScopes xmi:id="ManagementScope_1" scopeName="
(cell):myhostNode01Cell:(node):myhostNode01" scopeType="node"/>
```

Any configuration objects that are stored in the security.xml file whose management scopes are outside the current process scope are not loaded in the current process. Instead, the management scope is loaded by servers that are contained within the myhostNode01 node. Any application server that is on the specific node can view the keystore configuration.

When you list the contents of the key.p12 file to show the self-signed certificate, note that the common name (CN) of the distinguished name (DN) is the host name of the resident machine. This listing enables you to verify the host name by its URL connections. Additionally, you can verify the host name from a custom trust manager. For more information, see csec_sslx509certtrustdecisions.dita.

Contents of default keystore

The following sample code shows the contents of the default key.p12 file in a keytool list:

```
keytool -list -keystore c:\WebSphere\AppServer\profile\AppSrv01\profiles\config
\cells\myhostNode01Cell\nodes\myhostNode01\key.p12 -storepass myhostNode01Cell
-storetype PKCS12 -v
```

```
Alias name: default
Entry type: keyEntry
Owner: CN=myhost.austin.ibm.com, O=IBM, C=US
Issuer: CN=myhost.austin.ibm.com, O=IBM, C=US
Valid from: 10/18/05 4:06 PM until: 10/18/06 4:06 PM
Certificate fingerprint:
         SHA1: 33:6E:9E:10:65:04:CE:7A:6C:C3:B1:79:8B:9A:05:49:AC:E5:67:F3
```

The default alias name and the keyEntry entry type indicate that the private key is stored with the public key, which represents a complete personal certificate. The certificate is owned by CN=myhost.austin.ibm.com, O=IBM, C=US and it is issued by the same entity, which is self-signed. By default, the certificate is valid for one year from the date of creation.

Additionally, in some signer-exchange situations, the certificate fingerprint ensures that the sent certificate has not been modified. The fingerprint, which is a hash algorithm output for the certificate, is displayed by the WebSphere Application Server runtime during an automated signer exchange on the client side. The client fingerprint must match the fingerprint that is displayed on the server. The runtime typically uses the SHA1 hash algorithm to generate certificate fingerprints.

Default truststore

In the following sample code, the keystore object represents the default trust.p12 truststore. The truststore contains signer certificates that are necessary for making trust decisions:

```
<keyStores xmi:id="KeyStore_2" name="NodeDefaultTrustStore"
password="{xor}349dkckdd=" provider="IBMJCE" location="${USER_INSTALL_ROOT}
/config/cells/myhostNode01Cell/nodes/myhostNode01/trust.p12" type="PKCS12"
fileBased="true" hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

Contents of default truststore

The following sample code shows the contents of the default trust.p12 truststore in a keytool listing. For the sample self-signed certificate, the default_signer alias name and the trustedCertEntry entry type indicate that the certificate is the public key. The private key is not stored in this truststore.

```
keytool -list -keystore c:\WebSphere\AppServer\profile\AppSrv01\profiles\config
\cells\myhostNode01Cell\nodes\myhostNode01\trust.p12 -storepass myhostNode01Cell
-storetype PKCS12 -v
```

```
Alias name: default_signer
Entry type: trustedCertEntry

Owner: CN=myhost.austin.ibm.com, O=IBM, C=US
Issuer: CN=myhost.austin.ibm.com, O=IBM, C=US
```

```
Valid from: 10/18/05 4:06 PM until: 10/18/06 4:06 PM
Certificate fingerprint:
         SHA1: 33:6E:9E:10:65:04:CE:7A:6C:C3:B1:79:8B:9A:05:49:AC:E5:67:F3
```

Secure installation for client signer retrieval:

Each profile in the WebSphere Application Server environment contains a unique self-signed certificate that was created when the profile was created. This certificate replaces the default dummy certificate that ships with WebSphere Application Server in releases prior to version 6.1. When a profile is federated to a deployment manager, the signer for that self-signed certificate is added to the common truststore for the cell.

By default, clients do not trust servers from different profiles in the WebSphere Application Server environment. That is, they do not contain the signer for these servers. There are some things that you can do to assist in establishing this trust:

1. Enable the signer exchange prompt to except the signer during the connection attempt.
2. Run the **retrieveSigners** utility to download the signers from that system prior to making the connection.
3. Copy the `trust.p12` file from the `/config/cells/<cell_name>/nodes/<node_name>` directory of the server profile to the `/etc` directory of the client. Update the SSL configuration to reflect the new file name and password, if they are different. Copying the file provides the client with a `trust.p12` that contains all signers from servers in that cell. Also, you might need to perform this step for back-level clients that are still using the `DummyClientTrustFile.jks` file. In this case, you might need to change the `sas.client.props` or `soap.client.props` file to reflect the new truststore, truststore password, and truststore type (`PKCS12`).

For clients to perform an in-band signer exchange, you must specify the `ssl.client.props` file as a com.ibm.SSL.ConfigURL property in the SSL configuration. For managed clients, this is done automatically. Signers are designated either as in-band during the connection or out-of-band during runtime. You must also set the com.ibm.ssl.enableSignerExchangePrompt attribute to `true`.

Tip: You can configure a certificate expiration monitor to replace server certificates that are about to expire. For more information about how clients can retrieve the new signer from the configuration, see csec_sslcertmonitoring.dita.

Using the signer exchange prompt to retrieve signers from a client

When the client does not already have a signer to connect to a process, you can enable the signer exchange prompt if you want to be able to accept the signer during the connection attempt. The prompt displays once for each unique certificate and for each node. After the signer for the node is added, the signer remains in the client truststore. The following sample code shows the signer exchange prompt retrieving a signer from a client:

```
C:\WASX_e0540.11\AppServer\profiles\AppSrv01\bin\serverStatus -all
ADMU0116I: Tool information is being logged in file
           C:\WASX_e0540.11\AppServer\profiles\AppSrv01\logs\serverStatus.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU0503I: Retrieving server status for all servers
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr

*** SSL SIGNER EXCHANGE PROMPT ***
SSL signer from target host 192.168.1.5 is not found in truststore
C:\WebSphere\AppServer\profiles\AppSrv01\etc\trust.p12.

Here is the signer information (verify the digest value matches what is
displayed at the server):

Subject DN:    CN=myhost.austin.ibm.com, O=IBM, C=US
```

```
Issuer DN:     CN=myhost.austin.ibm.com, O=IBM, C=US
Serial number: 1128544457
Expires:       Thu Oct 05 15:34:17 CDT 2006
SHA-1 Digest:  91:A1:A9:2D:F2:7D:70:0F:04:06:73:A3:B4:A4:9C:56:9D:A8:A3:BA
MD5 Digest:    88:72:C5:88:00:1C:A7:FA:D6:EB:04:88:AC:A1:C9:13

Add signer to the truststore now? (y/n) y
A retry of the request may need to occur.
ADMU0508I: The Deployment Manager "dmgr" is STARTED
```

To automate this process, see rxml_retrievesigners.dita.

When a prompt occurs to accept the signer, a socket timeout can occur and the connection might be broken. For this reason, the message `A retry of the request may need to occur.` displays after answering the prompt. The message informs the user to resubmit the request. This problem should not happen frequently, and it might be more prevalent for some protocols than others.

Verify the displayed SHA-1 digest, which is the signature of the certificate that is sent by the server. If you look at the certificate on the server, verify that the same SHA-1 digest displays.

You can disable the prompt when you do not want it to display by running the **retrieveSigners** utility to retrieve all of the signers for a particular cell. You can download or upload the signers from any remote keystore to any local keystore by referencing a common truststore with this client script. For more information, see csec_ssldefselfsigncertconf.dita.

Using the retrieveSigners utility to download signers for a client

You can run the **retrieveSigners** utility to retrieve all of the signers from the remote keystore for a specified client keystore. The truststore contains the signers that enable the client to connect to its processes. The **retrieveSigners** utility can point to any keystore in the target configuration, within the scope of the target process, and can download the signers (certificate entries only) to any client keystore in the `ssl.client.props` file.

Obtaining signers for clients and servers from a previous release

When a client from a release prior to version 6.1 connects to the current release, the client must obtain signers for a successful handshake. Clients using previous releases of WebSphere Application Server cannot obtain signers as easily as in the current release. You can copy the deployment manager *common* truststore to your back-level client or server, and then re-configure the SSL configuration to directly reference that truststore. This common truststore of type `PKCS12` is located in the `/config/cells/<cell_name>/nodes/<node_name>` directory in the configuration repository and has a default password of `WebAS`.

To collect all of the signers for the cell in a single `trust.p12` keystore file, complete following steps:

1. Copy the trust.p12 keystore file on the server and replicate it on the client. The client references the file directly from the `sas.client.props` and `soap.client.props` files that specify the SSL properties for previous releases.
2. Change the client-side keystore password so that it matches the default cell name that is associated with the copied keystore.
3. Change the default keystore type for the `trust.p12` file to `PKCS12` in the client configuration.

The following two code samples show you a before and an after view of the changes to make.

Default SSL configuration of sas.client.props for a previous release

```
com.ibm.ssl.protocol=SSL
com.ibm.ssl.keyStore=file\:///
C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/DummyClientKeyFile.jks
```

```
com.ibm.ssl.keyStorePassword={xor}CDo9Hgw\=
com.ibm.ssl.keyStoreType=JKS
com.ibm.ssl.trustStore=file\:///
C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/DummyClientTrustFile.jks
com.ibm.ssl.trustStorePassword={xor}CDo9Hgw\=
com.ibm.ssl.trustStoreType=JKS
```

SSL configuration changes that are required to common truststore file in the /etc directory of the client

```
com.ibm.ssl.protocol=SSL
com.ibm.ssl.keyStore=file\:///
C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/DummyClientKeyFile.jks
com.ibm.ssl.keyStorePassword={xor}CDo9Hgw\=
com.ibm.ssl.keyStoreType=JKS
com.ibm.ssl.trustStore=file\:///
C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/trust.p12
com.ibm.ssl.trustStorePassword=myhostNode01Cell
com.ibm.ssl.trustStoreType=PKCS12
```

You can also make these changes in the `soap.client.props` file and specify the `key.p12` file in place of the `DummyClientKeyFile.jks` file. However, you must also change the keyStorePassword and keyStoreType values to match those in the default `key.p12` file.

In releases of WebSphere Application Server prior to version 6.1, you must edit the SSL configuration on the server to replace the common truststore. The `trust.p12` file, which is used by the server, also must contain the default dummy certificate signer for connections among servers at previous release levels. You might need to manually extract the default certificate from the `DummyServerKeyFile.jks` file and hen import the certificate into the `trust.p12` file that you added to the configuration.

retrieveSigners command: The **retrieveSigners** command creates a new client self-signed certificate, keystore, and SSL configuration in the ssl.client.props file. Using this command you can optionally extract the signer to a file.

For more information about where to run this command, see the Using command tools article.

Syntax

The command syntax is as follows:

```
retrieveSigners <remoteKeyStoreName> <localKeyStoreName> [options]
```

where the `<remoteKeyStoreName>` and `<localKeyStoreName>` parameters are required. The optional parameters include the following:

```
[-remoteAlias aliasFromRemoteStore]
[-localAlias storeAsAlias]
[-listRemoteKeyStoreNames][-listLocalKeyStoreNames]
[-autoAcceptBootstrapSigner][-uploadSigners] [-host host]
[-port port][-conntype RMI|SOAP][-user user]
[-password password]
[-trace] [-logfile filename]
[-replacelog] [-quiet] [-help]
```

Parameters

The following parameters are available for the **retrieveSigners** command:

-remoteKeyStoreName
The name of a truststore that is located in the server configuration from which to retrieve the signers. This will typically be the CellDefaultTrustStore file for an managed environment or the NodeDefaultTrustStore file for an unmanaged environment.

-localKeyStoreName
The name of the truststore that is located in the ssl.client.props file for the profile to which the retrieved signers is added. This will typically be the ClientDefaultTrustStore file for either a managed or unmanaged environment.

-remoteAlias <aliasFromRemoteStore>
Specifies one alias from the remote truststore that you want to retrieve. Otherwise, all signers from the remote truststore will be retrieved.

-localAlias <storeAsAlias>
Determines the name of the alias stored in the local truststore. This option is only valid if you specify the –remoteAlias option. If you do not specify the -localAlias option, the alias name from the remote truststore will be used, if possible. If an alias clash occurs, the alias name will be used and it will have an incremented number appended to the end of it until it finds a unique alias.

-listRemoteKeyStoreNames
Sends a remote request to the server to list all keystores that you can specify for the remoteKeyStoreName parameter. Use this command when you are unsure of the name of the remote truststore that you want to download the signers from.

-listLocalKeyStoreNames
Lists the keystores located in the ssl.client.props file that you can specify for the localKeyStoreName parameter. This truststore will receive the signers from the server. Use this parameter when you are unsure of the name of the local truststore that you want to retrieve the signers into. The default name of the truststore is ClientDefaultTrustStore and is located in the ssl.client.props file.

-autoAcceptBootstrapSigner
Automatically adds a signer in order to make a secure connection to the server. The purpose of the option is to allow automation of the command so that you do not need to accept the signer. After the signer is added to the local truststore, a SHA hash will print so that you can verify the certificate.

-uploadSigners
Converts the signer download into a signer upload. The signers from the localKeyStoreName parameter will be sent to the remoteKeyStoreName parameter instead.

-host <host>
Specifies the target host from which the signers will be retrieved.

-port <port>
Specifies the target administrative port to which to connect. You must specify the port based on the -conntype parameter. If the conntype is SOAP, the default port is 8879. This can vary for different servers. If the conntype is RMI, the default port is 2809. This can vary for different servers.

-conntype <RMI|Soap>
Determines the administrative connector type that is used for the MBean call to retrieve the signers.

-user <user>
When global security is enabled, you can specify this option to supply the user name that will be authenticated for the MBean operation. This must be an identity with administrator authority. If you do not specify this parameter when global security is enabled, you will be prompted for credentials by default.

-password <password>
When global security is enabled, you can specify this option to supply the password that will be authenticated for the MBean operation. The password goes along with the –user parameter.

-trace
When specified, this enables tracing of the trace specification necessary to debug this component. By default, the trace will appear in the profiles/*profile_name*/log/retrieveSigners.log. file.

-logfile <filename>
Overrides the default trace file. By default, the trace will appear in the profiles/*profile_name*/log/retrieveSigners.log. file.

-replacelog
Causes the existing trace file to be replaced when the command is executed.

-quite
Suppresses most messages from printing out on the console.

-help
Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

- The following example lists remote and local keystores:

```
retrieveSigners.bat -listRemoteKeyStoreNames -listLocalKeyStoreNames -conntype RMI -port 2809 [Windows]

retrieveSigners.sh -listRemoteKeyStoreNames -listLocalKeyStoreNames -conntype RMI -port 2809 [Unix]
```

  Example output

```
CWPKI0306I: The following remote keystores exist on the specified server:
           CMSKeyStore, NodeLTPAKeys, NodeDefaultTrustStore, NodeDefaultKeyStore
CWPKI0307I: The following local keystores exist on the client:
           ClientDefaultKeyStore, ClientDefaultTrustStore
```

- The following example retrieves all signers from NodeDefaultTrustStore:

```
retrieveSigners.bat NodeDefault TrustStore ClientDefaultTrustStore -autoAcceptBootstrapSigner
-conntype RMI -port 2809 [Windows]

retrieveSigners.sh NodeDefault TrustStore ClientDefaultTrustStore -autoAcceptBootstrapSigner
-conntype RMI -port 2809 [Unix]
```

  Example output

```
CWPKI0308I: Adding signer alias "CN=BIRKT40.austin.ibm.com, O=IBM, C=US" to
           local keystore "ClientDefaultTrustStore" with the following SHA
           digest: 40:20:CF:BE:B4:B2:9C:F0:96:4D:EE:E5:14:92:9E:37:8D:51:A5:47
```

Certificate expiration monitoring:

The certificate expiration monitor administrative task cycles through all the keystores that are configured in the `security.xml` file and reports on any certificates that expire within a specified threshold, which is typically within 30 days.

The default self-signed certificate on each node expires 365 days after creation. You can modify the certificate validity period by changing the default value for the `com.ibm.ssl.defaultCertReqDays=365` property in the `ssl.client.props` global property area for clients. You can also specify this property as a security custom property on the administrative console. Click **Security > Secure administration, applications, and infrastructure > Custom properties**.

You can configure this monitor task to run according to a particular schedule. The schedule produces the next start date that persists in the configuration and, when the date is reached, WebSphere Application Server starts the monitor to check all of the keystores for certificates that meet the expiration threshold. You can start the task manually to run at any time.

The following `security.xml` configuration object specifies when the monitor task starts, determines the certificate expiration threshold, and indicates whether you are notified in an e-mail using Simple Mail Transfer Protocol (SMTP) or in a message log.

```
<wsCertificateExpirationMonitor xmi:id="WSCertificateExpirationMonitor_1"
name="Certificate Expiration Monitor" daysBeforeNotification="30"
isEnabled="true" autoReplace="true" deleteOld="true"
wsNotification="WSNotification_1" wsSchedule="WSSchedule_2"
nextStartDate="1134358204849"/>
```

The expiration monitor automatically replaces only self-signed certificates that meet the expiration threshold criteria. To replace all of the signers from the old certificate with the signer that belongs to the new certificate in all the keystores in the configuration for that cell, set the autoReplace attribute to `true`. When the deleteOld attribute is `true`, the old personal certificate and old signers also are deleted from the keystores. The isEnabled attribute determines whether the expiration monitor task runs based upon the nextStartDate attribute that is derived from the schedule. The nextStartDate attribute is derived from the schedule in milliseconds since 1970, and is identical to the System.currentTimeMillis(). If the nextStartDate has already passed when an expiration monitor process begins, and the expiration monitor is enabled, the task is started, but a new nextStartDate value is established based on the schedule.

The following sample code shows the frequency attribute as the number of days between each run.

```
<wsSchedules xmi:id="WSSchedule_2" name="ExpirationMonitorSchedule"
frequency="30" dayOfWeek="1" hour="21" minute="30"/>
```

The dayOfWeek attribute adjusts the schedule to run on a specified day of the week, which is always the same day regardless of whether the frequency is set to 30 or 31 days. Based on 24-hour clock, the hour and minute attributes determine when the expiration monitor is started on the specified day.

The following sample code shows the notification configuration, which notifies you after the expiration monitor runs.

```
<>wsNotifications xmi:id="WSNotification_1" name="MessageLog" logToSystemOut="true" emailList=""/
```

For expiration monitor notifications, you can select message log, e-mail using SMTP server, or both methods of notification. When you configure the e-mail option, use the format `user@domain@smtpserver`. If you do not specify an SMTP server, WebSphere Application Server defaults to the same domain as the e-mail address. For example, if you configure joeuser@ibm.com, WebSphere Application Server attempts to call smtp-server.ibm.com. To specify multiple e-mail addresses using scripting, you must add a pipe (|) character between entries. When you specify the logToSystemOut attribute, the expiration monitor results are sent to the message log for the environment, which is typically the `SystemOut.log` file.

Web server plug-in default configuration:

When you create a new Web server definition, WebSphere Application Server associates the Web server plug-in with a Certificate Management Services (CMS) keystore for a specific node. The keystore contains all of the signers for the current cell with the self-signed certificate, which belongs to the node. The plug-in can communicate securely to WebSphere Application Server, even when the plug-in is configured with Secure Sockets Layer (SSL) client authentication enabled.

When you set the Web server definition to webserver1 on node myhostNode01, WebSphere Application Server creates the keystore configuration. The keystore is scoped to the webserver1 server, which makes it visible to this server only. Other processes cannot use this keystore definition.

The following sample code from the `security.xml` file shows the configuration entries for the Web server plug-in.

```
<keyStores xmi:id="KeyStore_1132357815719" name="CMSKeyStore"
password="{xor}HRYNFAtrbxEwOzpvbhw6MzM=" provider="IBMCMSProvider"
location="C:\WASX_e0540.11\AppServer\profiles\AppSrv01/config/cells
/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.kdb"
type="CMSKS" fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132357815718"/>
<managementScopes xmi:id="ManagementScope_1132357815718" scopeName="
(cell):myhostCell01:(node):myhostNode01:(server):webserver1" scopeType="server"/>
```

The following sample code shows how the CMS keystore and stash file are generated in the security.xml file.

```
C:\WebSphere\AppServer\profiles\Dmgr01\config\cells\myhostCell01\nodes
\myhostNode01\servers\webserver1\plugin-key.kdb
C:\WebSphere\AppServer\profiles\Dmgr01\config\cells\myhostCell01
\nodes\myhostNode01\servers\webserver1\plugin-key.sth
```

The default password for the keystore is WebAS. You can change the default keystore password by using either the administrative console or the appropriate **AdminTask** command. The following sample code shows the **AdminTask** command that you can use to create this CMS keystore.

```
$AdminTask createCMSKeyStore /config/cells/myhostCell01/nodes/myhostNode01
/servers/webserver1/plugin-key.kdb myhost.austin.ibm.com
```

Note the following characteristics of the previous example:

- You can create only one CMSKeyStore entry for each management scope. If a CMS keystore already exists for scope (cell):myhostCell01:(node):myhostNode01:(server):webserver1, then you cannot create another CMSKeyStore entry
- The Uniform Resource Identifier (URI) for the keystore name is /config/cells/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.kdb
- The host name in the plug-in location is myhost.austin.ibm.com. WebSphere Application Server uses this name to create a self-signed certificate, if a self-signed certificate does not already exist for that particular node. If a self-signed certificate already exists for the node, then the certificate is put into the CMS keystore and all the signers from the cell are added, by default.

When additional nodes are federated, the signers for these nodes are not automatically added to each Web server for the CMS keystore. For the Web server plug-in to be able to communicate with a newly federated node, you must manually exchange signers with the CMSKeyStore keystore. Use the administrative console keystore certificate management function to exchange signers. For more information, see “Extracting a signer certificate from a personal certificate” on page 1485.

Dynamic configuration updates

During the Secure Sockets Layer (SSL) runtime, dynamic configuration updates affect both inbound and outbound SSL endpoints. For inbound SSL endpoints, the changes that are implemented by the SSL channel are only affected by dynamic changes. For outbound SSL endpoints, all outbound connections inherit the new configuration changes.

In this release, dynamic update functionality provides you with greater flexibility and efficiency. You can change SSL configurations without restarting WebSphere Application Server for the changes to take effect.

To make dynamic changes, in the administrative console click **Security > SSL certificates and key management**, then select the **Dynamically update the runtime when SSL configuration changes occur** check box. You must save your changes and then synchronize the security.xml file with remote systems. A remote system must be able to confirm that dynamicallyUpdateSSLConfig=true is in the security.xml file.

The SSL runtime reloads the modified SSL configuration and creates a new SSLEngine for the modified connections that are associated with inbound endpoints. New outbound connections use the new configuration while existing connections continue to use the old SSLEngine object and are not affected.

Tip: Make dynamic changes to the SSL configuration during off-peak hours. Synchronization delays can negatively affect connections when you update SSL configurations during peak hours.

You can turn on and off the dynamicallyUpdateSSLConfig attribute in the security.xml file to ensure successful updates by doing the following actions:

1. Set dynamicallyUpdateSSLConfig=On.

2. Save the updated configuration.
3. Synchronize the `security.xml` file with remote systems.
4. Set the dynamicallyUpdateSSLConfig attribute to `Off`.

You must verify that all of the nodes receive the changes before turning off the dynamicallyUpdateSSLConfig attribute. Test the changes in a test environment before updating the production environment.

Tip: Some SSL changes, especially administrative SSL changes, can cause server outages if you fail to test them first. When a change prevents trust between two endpoints, the endpoints cannot communicate with each other. Additionally, if administrative SSL connection updates cause system outages, you might need to disable the nodes after you make corrective changes using the deployment manager. From the command line, you can manually synchronize the server to retrieve the new SSL changes, then restart the nodes.

Restriction: In this release, the Object Request Broker (ORB) inbound SSL socket factory and the Admin SOAP inbound SSL socket factory are not affected by dynamic configuration changes. SSL socket factories cannot be bound to a port again without affecting the existing connections, with the exception of the SSL channel. You must restart the server to make a change to the SSL configuration for these two inbound protocols. Outbound configurations can still be changed dynamically. These changes are affected because the sockets are created during each new outbound connection.

Certificate management using iKeyman

Starting in WebSphere Application Server Version 6.1, you can manage your certificates from the administrative console. When using versions of WebSphere Application Server prior to Version 6.1, use iKeyman for certificate management. iKeyman is a key management utility.

WebSphere Application Server certificate management requires that you define the keystores in your WebSphere Application Server configuration. With iKeyman, you need access to the keystore file only. You can read a keystore file with personal certificates and signers that is created in iKeyman can be read into the WebSphere Application Server configuration by using the **createKeyStore** command.

The majority of certificate management functions are the same between WebSphere Application Server and iKeyman, especially for personal certificates and signer certificates. However, certificate requests are special. The underlying behavior is different in the two certificate management schemes. Because of the different behavior, when a certificate request is generated from iKeyman, the process must be completed in iKeyman. For example, a certificate that is generated by a certificate request that originated in iKeyman must be received in iKeyman as well.

The same is true for WebSphere Application Server. For example, when a certificate is generated from a certificate request that originated in WebSphere Application Server, the certificate must be received in WebSphere Application Server.

You can perform the following certificate operations using iKeyman:

Types of certificates	Functions	Description
Personal certificates	Create a self-signed certificate	Creates a self-signed certificate and stores it in a keystore.
	List personal certificates	Lists all the personal certificates in a keystore.
	Get information about a personal certificate	Gets information about a personal certificate.
	Delete a personal certificate	Deletes a personal certificate from a keystore.
	Import a certificate	Imports a certificate from a keystore to a keystore.

Types of certificates	Functions	Description
	Export a certificate	Exports a certificate from a keystore to another keystore.
	Extract a certificate	Extracts the signer part of a personal certificate to a file.
	Receive a certificate	Reads a certificate that comes from a certificate authority (CA) into a keystore.
Signer certificates	Add a signer certificate	Adds a signer certificate from a file to a keystore.
	List signer certificates	Lists all the signer certificates in a keystore.
	Get information about a signer certificate	Gets information about a signer certificate.
	Delete a signer certificate	Deletes a signer certificate from a keystore.
	Extract a signer certificate	Extracts a signer certificate from a keystore, and stores the certificate in a file.
Certificate requests	Create a certificate request	Creates a certificate request that can be sent to a CA.
	List certificate requests	Lists the certificate requests in a keystore.
	Get information about a certificate request	Gets information about a certificate request.
	Delete a certificate request	Deletes a certificate request from a keystore.
	Extract a certificate request	Extracts a certificate request to a file.

Certificate management

You can manage certificate operations that involve personal certificates, signer certificates, and personal certificate requests on the administrative console.

Types of certificates

WebSphere Application Server uses the certificates that reside in keystores to establish trust for a Secure Sockets Layer (SSL) connection. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name* > Key stores and certificates**, then select an existing or create a new keystore. After selecting a keystore, and depending on the type of certificate you need, choose one of the following types of certificates under Related Items:

- Personal certificate
- Signer certificate
- Personal certificate request

The following table describes the certificate operations that you can perform on the administrative console:

Types of certificates	Functions	Description
Personal certificates	Create a self-signed certificate	Creates a self-signed certificate and stores it in a keystore.
	List personal certificates	Lists all the personal certificates in a keystore.
	Get information about a personal certificate	Gets information about a personal certificate.
	Delete a personal certificate	Deletes a personal certificate from a keystore.
	Import a certificate	Imports a certificate from a keystore to a keystore.

Types of certificates	Functions	Description
	Export a certificate	Exports a certificate from a keystore to another keystore.
	Extract a certificate	Extracts the signer part of a personal certificate to a file.
	Exchange signer certificates	Exchange signer part of a personal certificate between key store.
	Receive a certificate	Reads a certificate that comes from a certificate authority (CA) into a keystore.
	Replace a certificate	Replaces all occurrences of a personal certificate alias in the WebSphere Application Server configuration with another certificate. Also, replaces all occurrences of the personal certificates signer with the new personal certificate signer.
Signer certificates	Add a signer certificate	Adds a signer certificate from a file to a keystore.
	List signer certificates	Lists all the signer certificates in a keystore.
	Get information about a signer certificate	Gets information about a signer certificate.
	Delete a signer certificate	Deletes a signer certificate from a keystore.
	Extract a signer certificate	Extracts a signer certificate from a keystore, and stores the certificate in a file.
	Retrieve a signer from a port	Retrieves a signer certificate from a port, and stores it in a key store.
Certificate requests	Create a certificate request	Creates a certificate request that can be sent to a CA.
	List certificate requests	Lists the certificate requests in a keystore.
	Get information about a certificate request	Gets information about a certificate request.
	Delete a certificate request	Deletes a certificate request from a keystore.
	Extract a certificate request	Extracts a certificate request to a file.

Personal certificates

The following table lists the operations that you can perform on personal certificates, the AdminTask object that you can use to perform that operation, and how to navigate to the certificate on the console:

Function	AdminTask object	Administrative console
Create a self-signed certificate	createSelfSigneCertificate	**Security** > **Secure Communications** > **Key store and certificates** > *key store* > **Create a Self-Signed Certificate**
List personal certificates	listPersonalCertificates	**Security** > **Secure Communications** > **Key store and certificates** > *key store* > **personal certificates**
Get information about a personal certificate	getPersonalCertificate	**Security** > **Secure Communications** > **Key store and certificates** > *key store* > **personal certificates** > *alias*
Delete a personal certificate	deletePersonalCertificate	**Security** > **Secure Communications** > **Key store and certificates** > *key store* > **personal certificates** > **delete**

Function	AdminTask object	Administrative console
Import a certificate	importCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates > import**
Export a certificate	exportCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates > export**
Extract a certificate	extractCertificate	**Security > Secure Communications > Key store and certificates >** *key store* **> personal certificates > extract**
Exchange signer certificates	exchangeSignerCertificates	**Security > Secure Communications > Key store and certificates > Exchange signers**

Signer certificates

The following table lists the operations that you can perform with signer certificates, the AdminTask object that you can use to perform the operation, and how to navigate to the certificate on the console:

Function	AdminTask object	Administrative console
Add a signer certificate	addSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates > Add**
List signer certificates	listSignerCertificates	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates**
Get information about a signer certificate	getSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates >** *alias*
Delete a signer certificate	deleteSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificate >delete**
Extract a signer certificate to a file	extractSignerCertificate	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates > extract**
Retrieve a signer certificate from a port	retrieveSignerFromPort	**Security > Secure communications > Key store and certificates >** *key store* **> signer certificates > retrieve from port**

Personal certificate requests

The following table lists the operations that you can perform on personal certificate requests, the AdminTask object that you can use to perform that operation, and how to navigate to the certificate request on the console:

Function	AdminTask object	Administrative console
Create a personal certificate request	createCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate Requests > Add**
List personal certificate requests	listCertificateRequests	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests**
Get information about a personal certificate request	getCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests >** *alias*

Function	AdminTask object	Administrative console
Delete a personal certificate request	deleteCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests > delete**
Extract a personal certificate request to a file	extractCertificateRequest	**Security > Secure communications > Key store and certificates >** *key store* **> Personal certificate requests > Extract**

Creating a Secure Sockets Layer configuration

Secure Sockets Layer (SSL) configurations contain the attributes that you need to control the behavior of client and server SSL endpoints. You create SSL configurations with unique names within specific management scopes on the inbound and outbound tree in the configuration topology. This task shows you how to define SSL configurations, including quality of protection and trust and key manager settings.

You must decide at which scope you need to define an SSL configuration, for instance, the cell, node group, node, server, cluster, or endpoint scope, from the least specific to the most specific scope. When you define an SSL configuration at the node scope, for example, only those processes within that node can load the SSL configuration; however, any processes at the endpoint in the cell can use an SSL configuration at the cell scope, which is higher in the topology.

You must also decide which scope to associate with the new SSL configuration, according to the processes that the configuration affects. For example, an SSL configuration for a hardware cryptographic device might require a keystore that is available only on a specific node, or you might need an SSL configuration for a connection to a particular SSL host and port. For more information, see “Dynamic outbound selection of Secure Sockets Layer configurations” on page 1409.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations**.
2. Select an SSL configuration link on either the Inbound or Outbound tree, depending on the process you are configuring.
   - If the scope is already associated with a configuration and alias, the SSL configuration alias and certificate alias are noted in parentheses.
   - If the parenthetical information is not included, then the scope is not associated. Instead, the scope inherits the configuration properties of the first scope above it that is associated with an SSL configuration and certificate alias.

   The cell scope must be associated with an SSL configuration because it is at the top of the topology and represents the default SSL configuration for the inbound or outbound connection.
3. Click **SSL configurations** under Related Items. You can view and select any of the SSL configurations that are configured at this scope. You can also view and select these configuration at every scope that is lower on the topology.
4. Click **New** to display the SSL configuration panel. You cannot select links under Additional Properties until you type a configuration name and click **Apply**.
5. Type an SSL configuration name. This field is required. The configuration name is the SSL configuration alias. Make the alias name unique within the list of SSL configuration aliases that are already created at the selected scope. The new SSL configuration uses this alias for other configuration tasks.
6. Select a truststore name from the drop-down list. A truststore name refers to a specific truststore that holds signer certificates that validate the trust of certificates sent by remote connections during an

SSL handshake. If there is no truststore in the list, see “Creating a keystore configuration” on page 1466 to create a new truststore, which is a keystore whose role is to establish trust during the connection.

7. Select a keystore name from the drop-down list. A keystore contains the personal certificates that represent a signer identity and the private key that WebSphere Application Server uses to encrypt and sign data.
   - If you change the keystore name, click **Get certificate aliases** to refresh the list of certificates from which you can choose a default alias. WebSphere Application Server uses a server alias for inbound connections and a client alias for outbound connections.
   - If there is no keystore in the list, see “Creating a keystore configuration” on page 1466 to create a new keystore.
8. Choose a default server certificate alias for inbound connections. Select the default only when you have not specified an SSL configuration alias elsewhere and have not selected a certificate alias. A centrally managed SSL configuration tree can override the default alias. For more information, see “Central management of Secure Sockets Layer configurations” on page 1410.
9. Choose a default client certificate alias for outbound connections. Select the default only when the server SSL configuration specifies an SSL client authentication.
10. Review the identified management scope for the SSL configuration. Make the management scope in this field identical to the link you selected in Step 2. If you want to change the scope, you must click a different link in the topology tree and continue at Step 3.
11. Click **Apply** if you intend to configure Additional Properties. If not, go to Step 24.
12. Click **Quality of protection (QoP) settings** under Additional Properties. QoP settings define the strength of the SSL encryption, the integrity of the signer, and the authenticity of the certificate.
13. Select a client authentication setting to establish an SSL configuration for inbound connections and for clients to send their certificates, if appropriate.
    - If you select **None**, the server does not request that a client send a certificate during the handshake.
    - If you select **Supported**, the server requests that a client send a certificate. However, if the client does not have a certificate, the handshake might still succeed.
    - If you select **Required**, the server requests that a client send a certificate. However, if the client does not have a certificate, the handshake fails.

    Important: The signer certificate that represents the client must be in the truststore that you select for the SSL configuration. By default, servers within the same cell trust each other because they use the common truststore, `trust.p12`, that is located in the cell directory of the configuration repository. However, if you use keystores and truststores that you create, perform a signer exchange before you select either **Supported** or **Required**.
14. Select a protocol for the SSL handshake.
    - The default protocol, SSL_TLS, supports client protocols TLSv1, SSLv3, and SSLv2.
    - The TLSv1 protocol supports TLS and TLSv1. The SSL server connection must support this protocol for the handshake to proceed.
    - The SSLv3 protocol supports SSL and SSLv3. The SSL server connection must support this protocol for the handshake to proceed.

    Important: Do not use the SSLv2 protocol for the SSL server connection. Use it only when necessary on the client side.
15. Select one of the following options:
    - A predefined Java Secure Socket Extension (JSSE) provider. The IBMJSSE2 provider is recommended for use on all platforms which support it. It is required for use by the channel framework SSL channel. When Federal Information Processing Standard (FIPS) is enabled, IBMJSSE2 is used in combination with the IBMJCEFIPS crypto provider.

- A custom JSSE provider. Type a provider name in the **Custom provider** field.

16. Select from among the following cipher suite groups:
    - **Strong**: WebSphere Application Server can perform 128-bit confidentiality algorithms for encryption and support integrity signing algorithms. However, a strong cipher suite can affect the performance of the connection.
    - **Medium**: WebSphere Application Server can perform 40-bit encryption algorithms for encryption and support integrity signing algorithms.
    - **Weak**: WebSphere Application Server can support integrity signing algorithms but not to perform encryption. Select this option with care because passwords and other sensitive information that cross the network are visible to an Internet Protocol (IP) sniffer.
    - **Custom**: you can select specific ciphers. Any time you change the ciphers that are listed from a specific cipher suite group, the group name changes to Custom.
17. Click **Update selected ciphers** to view a list of the available ciphers for each cipher strength.
18. Click **OK** to return to the new SSL configuration panel.
19. Click **Trust and key managers** under Additional Properties.
20. Select a default trust manager for the primary SSL handshake trust decision.
    - Choose **IbmPKIX** when you require certificate revocation list (CRL) checking using CRL distribution points in the certificates.
    - Choose **IbmX509** when you do not require CRL checking but do need increased performance. You can configure a custom trust manager to perform CRL checking, if necessary.
21. Define a custom trust manager, if appropriate. You can define a custom trust manager that runs with the default trust manager you select. The custom trust manager must implement the JSSE javax.net.ssl.X509TrustManager interface and, optionally, the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface to obtain product-specific information.
    a. Click **Security > SSL certificate and key management > Manage endpoint security configurations > *SSL_configuration* > Trust and key managers > Trust managers > New**.
    b. Type a unique trust manager name.
    c. Select the **Custom** option.
    d. Type a class name.
    e. Click **OK**. When you return to the Trust and key managers panel, the new custom trust manager displays in the **Additional ordered trust managers** field. Use the left and right list boxes to add and remove custom trust managers.
22. Select a key manager for the SSL configuration. By default, IbmX509 is the only key manager unless you create a custom key manager.

    Important: If you choose to implement your own key manager, you can affect the alias selection behavior because the key manager is responsible for selecting the certificate alias from the keystore. The custom key manager might not interpret the SSL configuration as the WebSphere Application Server key manager IbmX509 does. To define a custom key manager, click **Security > Secure communications > SSL configurations > *SSL_configuration* > Trust and key managers > Key managers > New**.
23. Click **OK** to save the trust and key manager settings and return to the new SSL configuration panel.
24. Click **Save** to save the new SSL configuration.

Important: You can override the default trust manager when you configure at least one custom trust manager and set the com.ibm.ssl.skipDefaultTrustManagerWhenCustomDefined property to `true`. Click **Custom Property** on the SSL configuration panel. However, if you change the default, you leave all the trust decisions to the custom trust manager, which is not recommended for production environments. In test environments, use a dummy trust manager to avoid certificate validation. Remember that these environment are not secure.

In this release of WebSphere Application Server, you can associate SSL configurations with protocols using one of the following methods:

- Set the SSL configuration on the thread programmatically
- Associate the SSL configuration with an outbound protocol, and target host and port. For more information, see “Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint” on page 1447
- Associate the SSL configuration directly using the alias. For more information, see tsec_sslselconfigdirect.dita
- Manage the SSL configurations centrally by associating them with SSL configuration groups or zones that are scoped for endpoints. For more information, see “Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes” on page 1450

SSL certificate and key management

Use this page to configure security for Secure Socket Layer (SSL) and key management, certificates, and notifications. The SSL protocol provides secure communications between remote server processes or endpoints. SSL security can be used for establishing communications inbound to and outbound from an endpoint. To establish secure communications, a certificate and an SSL configuration must be specified for the endpoint.

To view this administrative console page, click **Security > SSL certificate and key management**.

Configuration settings:

Specifies the following administrative console tasks:

- Manage endpoint security configurations
- Manage certificate expiration

Use Federal Information Processing Standard (FIPS) algorithms:

Specifies the Federal Information Processing Standard (FIPS)-compliant Java cryptography engine is enabled.

- Does not affect the SSL cryptography that is performed by the application server for z/OS System Secure Sockets Layer (SSSL).
- Does not change the JSSE provider if this cell includes any Application Server versions before the application server for z/OS Version 6.0.x.

When you select the **Use the Federal Information Processing Standard (FIPS)** option, the Lightweight Third Party Authentication (LTPA) implementation uses IBMJCEFIPS. IBMJCEFIPS supports the Federal Information Processing Standard (FIPS)-approved cryptographic algorithms for Data Encryption Standard (DES), Triple DES, and Advanced Encryption Standard (AES). Although the LTPA keys are backwards compatible with prior releases of the application server, the LTPA token is not compatible with prior releases. In prior releases, the application server did not generate the LTPA token using a FIPS-approved algorithm.

The IBMJSSE2 JSSE provider does not perform cryptographic functions directly, and therefore does not need to be FIPS-approved. Instead, the IBMJSSE2 JSSE provider uses the JCE framework for cryptographic functions and uses IBMJCEFIPS when FIPS mode is enabled.

Important: HP-UX The IBMJSSEFIPS provider is not supported on the HP-UX platform. However, the IBMJSSE2 provider, which uses IBMJCEFIPS, is supported on the HP-UX platform.

Default: Disabled

Dynamically update the runtime when SSL configuration changes occur:

Specifies that all of the SSL-related attributes that change must be read from the configuration dynamically after they have been saved, then reused for new connections. To avoid customer impact, it is recommended that changes to production servers be made during off-peak periods.

Default: Disabled

When this option is selected, the configuration is updated each time you configure an SSL communication.

SSL configurations for selected scopes

Use this page to display Secure Socket Layer (SSL) configurations for selected scopes, such as a cell, node, server, or cluster. From this page you can navigate to configuration panels for the following: SSL configurations, dynamic inbound and outbound endpoint SSL configurations, key stores, key sets, key set groups, key managers, and trust managers.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***.

Name:

Specifies the SSL configuration scope, which is derived from the selected object in the hierarchy.

Data type: Text

Direction:

Specifies the direction for which the SSLConfig applies. Inbound refers to any listener port. Outbound refers to outbound end point connections.

Data type: Text

Inherited SSL configuration name:

Specifies the name of the SSL configuration that is inherited from a higher level scope

This field displays for server and nodegroups within the object hierarchy.

This field does not display for the top-level node.

Data type: Text

Inherited certificate alias:

Specifies the certificate alias that is inherited from a higher-level scope.

This field displays for server and node groups within the object hierarchy.

This field does not display for the top-level node.

Data type: Text

Override inherited values:

Specifies the SSL configuration to be used for this scope and any lower scopes that have not already designated an SSL configuration.

This field displays for server and node groups within the object hierarchy.

This field does not display for the top-level node.

Default: Disabled

SSL configuration:

Specifies the SSL configuration that is used by requests at this scope.

Data type: Text

Update certificate alias list:

Specifies the certificate aliases contained in the key store for this SSL configuration can be selected from the **Certificate alias in key store** list. You must update the certificate list after choosing a different SSL configuration alias. If you do not update the list, you will save a certificate alias that is not contained in the SSL configuration.

Manage certificates:

Specifies to open the keystore panel for the key store in this SSL configuration, which enables you to manage personal certificates, signers, and certificate requests.

Certificate alias in key store:

Specifies the certificate to use in the key store.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Data type: Text

SSL configurations collection

Use this page to define a list of Secure Sockets Layer (SSL) configurations.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **SSL configurations**.

Button	Resulting action
New	The Java Secure Socket Extension (JSSE) repertoire is for Java-based SSL communications. You can define a new JSSE configuration that can be used to create an SSLContext, URLStreamHandler, SSLSocketFactory, SSLServerSocketFactory, and so on, using the com.ibm.websphere.ssl.JSSEHelper API.
Delete	Deletes an existing JSSE configuration (administrator only). Be careful that any references to the SSL configuration have been removed prior to deleting this SSL configuration.

Name:

Specifies the unique name of the SSL configuration in the management scope.

SSL configuration settings

Use this page to define Secure Sockets Layer (SSL) configuration properties.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration_name***. Under **Related items** click **SSL configurations > New**.

Name:

Specifies the unique name of the SSL configuration within the management scope in which it resides. For ways to programmatically access the properties that are configured for this SSL configuration, see the com.ibm.websphere.ssl.JSSEHelper application programming interface (API).

Data type: Text

Trust store name:

Specifies a reference to a specific trust store used by Java Secure Sockets Extension (JSSE). The trust store holds signer certificates that validate the trust of certificates sent by remote connections during an SSL handshake.

Data type: Text
Default: *selected trust store*

Key store name:

Specifies a reference to a specific key store. The key store holds personal certificates that represent the identity of one side of a connection. The public key of this personal certificate is sent to the other side of the connection to establish trust during the handshake. The remote side of the connection needs the root certificate authority (CA) certificate or self-signed public key (signer) to be in the trust store to validate this personal certificate.

Data type: Text
Default: *selected key store*

Get certificate aliases:

Queries the keystore for the aliases of all the personal certificates in the keystore from which to choose.

Default server certificate alias:

Specifies the certificate alias used as the identity for this SSL configuration if one has not been specified elsewhere.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Data type: Text

Default client certificate alias:

Specifies the description for a client certificate alias.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Data type: Text

Management scope:

Specifies the scope where this SSL configuration is visible. For example, if you choose a specific node, then the configuration is visible only on that node and on any servers that are part of that node.

Data type: Text

Creating a custom trust manager configuration

You can create a custom trust manager configuration at any management scope and associate the new trust manager with a Secure Sockets Layer (SSL) configuration.

You must develop, package, and locate a Java Archive JAR file for a custom key manager in the `was.install.root/lib/ext` directory on WebSphere Application Server. For more information, see "Example: Developing a custom trust manager for custom SSL trust decisions" on page 1439.

Complete the following steps in the administrative console:

1. Decide whether you want to create the custom trust manager at the cell scope or below the cell scope at the node, server, or cluster, for example.

   Important: When you create a custom trust manager at a level below the cell scope, you can associate it only with a Secure Sockets Layer (SSL) configuration at the same scope or higher. An SSL configuration at a scope lower than the trust manager does not see the trust manager configuration.

   - To create a custom trust manager at the cell scope, click **Security > SSL certificate and key management > Trust managers**. Every SSL configuration in the cell can select the trust manager at the cell scope.
   - To create a custom trust manager at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Trust managers**.
2. Click **New** to create a new custom trust manager.
3. Type a unique trust manager name.
4. Select the **Custom** implementation setting. The custom setting enables you to define a Java class with an implementation of the javax.net.ssl.X509TrustManager Java interface and, optionally, the com.ibm.wsspi.ssl.TrustManagerExtendedInfo WebSphere Application Server interface.

   Note: The standard implementation setting applies only when the trust manager is already defined in the Java security provider list as a provider and an algorithm, which is not the case for a custom trust manager.
5. Type a class name, for example, com.ibm.test.CustomTrustManager.
6. Select one of the following actions:
   - Click **Apply**, then click **Custom properties** under Additional Properties to add custom properties to the new custom trust manager. When you are finished adding custom properties, click **OK** and **Save**, then go to the next step.

- Click **OK** and **Save**, then go to the next step.

7. Click **SSL certificate and key management** in the page navigation at the top of the panel.
8. Select one of the following actions:
   - Click **SSL configurations** under Related Items for a cell-scoped SSL configuration.
   - Click **Manage endpoint security configurations** to select an SSL configuration at a lower scope.
9. Click the link for the existing SSL configuration that you want to associate with the new custom trust manager. You can create a new SSL configuration instead of associating the custom trust manager with an existing configuration. For more information, see “Creating a Secure Sockets Layer configuration” on page 1429.
10. Click **Trust and Key managers** under Additional Properties. If the new custom trust manager is not listed in the **Additional ordered trust managers** list, verify that you selected an SSL configuration scope that is at the same level or below the scope that you selected in Step 8.
11. Click **Add**. This action adds the new trust manager to the list of custom trust managers.
12. Click **OK** and **Save**.

You have created a custom trust manager configuration that references a JAR file in the install directory of WebSphere Application Server and associates it with an SSL configuration during the connection handshake.

You can create a custom trust manager for a pure client. For more information, see “Commands for the TrustManagerCommands group of the AdminTask object” on page 1675.

Trust and key managers settings:

Use this page to specify trust and key managers for the selected SSL configuration.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound >** *SSL_configuration_name*. Under **Related items** click **SSL configurations >** *SSL_configuration_name* | **New**. Under **Additional Properties** click **Trust and key managers**.

Attention: The application server checks the default trust managers first before checking the additional ordered trust managers in descending order.

Default trust manager:

Specifies the default trust manager, which is typically the IbmX509 trust manager by the IBMJSSE2 provider. The other default trust manager is IbmPKIX, which can be selected when certificate revocation checks must be made using the X509Certificate CRL distribution list. The IbmPKIX trust manager does not perform as well as the IbmX509 trust manager.

Data type:	Text
Default:	ibmX509TrustManager

Additional ordered trust managers:

Specifies additional trust managers that are used in the order shown for this SSL configuration.

Add:

Specifies to add the selection to the **Additional ordered trust managers** right-hand list.

Remove:

Specifies to remove the selection from the **Additional ordered trust managers** right-hand list.

Key manager:

Specifies the key manager that runs for this SSL configuration.

Data type: Text
Default: IbmX509KeyManager

Trust managers collection:

Use this page to define the implementation settings for the trust manager. A trust manager is a class that is invoked during an Secure Sockets Layer (SSL) handshake to make trust decisions about the remote end point. A default trust manager is used to validate the signature and expiration of the certificate. Custom trust managers can be plugged in to perform an extended certificate and host name check.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Trust managers**.

Button	Resulting action
New	Adds a new trust manager that can be selected by an SSL configuration. A trust manager is invoked during an SSL handshake and can decide whether the handshake should be accepted based on the information it knows about the remote certificate and host.
Delete	Deletes an existing trust manager. Make sure the trust manager is not referenced by any SSL configuration before you delete it.

Name:

Specifies the name of the trust manager. This name is used as a selection in the SSL configuration panel.

Class name:

Specifies a class that implements the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustMangerExtendedInfo interface to get extended information about the connection. The class can use the information to verify the host name and so on.

Algorithm:

Specifies the algorithm name of the trust manager that is implemented by the selected provider.

Trust managers settings:

This page enables you to view and set definitions for trust manager implementation settings. A trust manager is a class that gets invoked during a Secure Sockets Layer (SSL) handshake to make trust decisions about the remote end point. A default trust manager is used to validate the signature and expiration of the certificate. Custom trust managers can be plugged in to perform an extended certificate and hostname check.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items click **Trust managers > New** .

Name:

Specifies the name of the trust manager.

Data type: Text
Default: ibmX509TrustManager

Standard:

Specifies that the trust manager selection is available from a Java provider that is installed in the java.security file. This provider might be shipped by the Java Secure Sockets Extension (JSSE) or might be a custom provider that implements the javax.net.ssl.X509TrustManager interface.

Default: Enabled

Provider:

Specifies the provider name that has an implementation of the javax.net.ssl.X509TrustManager interface. This provider is typically set to IBMJSSE2.

Enabled when **Standard** is selected.

Default IBMJCE

Algorithm:

Specifies the algorithm name of the trust manager implemented by the selected provider.

Enabled when **Standard** is selected.

Default ibmX509 or IbmPKIX
Range ibmX509, IbmPKIX

Custom:

Specifies that the trust manager selection is based on a custom implementation class that implements the javax.net.ssl.X509TrustManager interface and optionally the com.ibm.wsspi.ssl.TrustManagerExctendedInfo interface to obtain additional connection information that is not otherwise available.

Default: Disabled

Class name:

Specifies a class that implements the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustMangerExtendedInfo interface to get extended information about the connection. The class can use the information to verify the host name and so on.

Enabled when **Custom** is selected.

Data type: Text

Example: Developing a custom trust manager for custom SSL trust decisions:

The following example is of a sample custom trust manager. The custom trust manager makes no trust decisions but instead uses the information in the X.509 certificate that it references to make decisions.

After you build and package the custom trust manager, configure it either from the `ssl.client.props` file for a pure client or the SSLConfiguration TrustManager link in the administrative console. See “Trust manager control of X.509 certificate trust decisions” on page 1403 for more information about trust managers.

Note: This example should only be used as a sample, and is not supported.

```
import java.security.cert.X509Certificate;
import javax.net.ssl.*;
import com.ibm.wsspi.ssl.TrustManagerExtendedInfo;

public final class CustomTrustManager implements X509TrustManager,
TrustManagerExtendedInfo
{
    private static ThreadLocal threadLocStorage = new ThreadLocal();
    private java.util.Properties sslConfig = null;
    private java.util.Properties props = null;

    public CustomTrustManager()
    {
    }

    /**
     * Method called by WebSphere Application Server run time to set the target
     * host information and potentially other connection info in the future.
     * This needs to be set on ThreadLocal since the same trust manager can be
     * used by multiple connections.
     *
     * @param java.util.Map - Contains information about the connection.
     */
    public void setExtendedInfo(java.util.Map info)
    {
        threadLocStorage.set(info);
    }

    /**
     * Method called internally to retrieve information about the connection.
     *
     * @return java.util.Map - Contains information about the connection.
     */
    private java.util.Map getExtendedInfo()
    {
        return (java.util.Map) threadLocStorage.get();
    }

    /**
     * Method called by WebSphere Application Server run time to set the custom
     * properties.
     *
     * @param java.util.Properties - custom props
     */
    public void setCustomProperties(java.util.Properties customProps)
    {
        props = customProps;
    }

    /**
     * Method called internally to the custom properties set in the Trust Manager
     * configuration.
     *
     * @return java.util.Properties - information set in the configuration.
     */
    private java.util.Properties getCustomProperties()
    {
        return props;
    }
```

```
/**
 * Method called by WebSphere Application Server runtime to set the SSL
 * configuration properties being used for this connection.
 *
 * @param java.util.Properties - contains a property for the SSL configuration.
 */
public void setSSLConfig(java.util.Properties config)
{
    sslConfig = config;
}

/**
 * Method called by TrustManager to get access to the SSL configuration for
 * this connection.
 *
 * @return java.util.Properties
 */
public java.util.Properties getSSLConfig ()
{
    return sslConfig;
}

/**
 * Method called on the server-side for establishing trust with a client.
 * See API documentation for javax.net.ssl.X509TrustManager.
 */
public void checkClientTrusted(X509Certificate[] chain, String authType)
    throws java.security.cert.CertificateException
{
    for (int j=0; j<chain.length; j++)
    {
        System.out.println("Client certificate information:");
        System.out.println("  Subject DN: " + chain[j].getSubjectDN());
        System.out.println("  Issuer DN: " + chain[j].getIssuerDN());
        System.out.println("  Serial number: " + chain[j].getSerialNumber());
        System.out.println("");
    }
}


/**
 * Method called on the client-side for establishing trust with a server.
 * See API documentation for javax.net.ssl.X509TrustManager.
 */
public void checkServerTrusted(X509Certificate[] chain, String authType)
    throws java.security.cert.CertificateException
{
    for (int j=0; j<chain.length; j++)
    {
        System.out.println("Server certificate information:");
        System.out.println("  Subject DN: " + chain[j].getSubjectDN());
        System.out.println("  Issuer DN: " + chain[j].getIssuerDN());
        System.out.println("  Serial number: " + chain[j].getSerialNumber());
        System.out.println("");
    }
}

/**
 * Return an array of certificate authority certificates which are trusted
 * for authenticating peers. You can return null here since the IbmX509
 * or IbmPKIX will provide a default set of issuers.
 *
 * See API documentation for javax.net.ssl.X509TrustManager.
 */
public X509Certificate[] getAcceptedIssuers()
```

```
    {
        return null;
    }
}
```

Related concepts

"Trust manager control of X.509 certificate trust decisions" on page 1403
The role of the trust manager is to validate the Secure Sockets Layer (SSL) certificate that is sent by the peer, which includes verifying the signature and checking the expiration date of the certificate. A Java Secure Socket Extension (JSSE) trust manager determines if the remote peer can be trusted during an SSL handshake.

Creating a custom key manager

You can create a custom key manager configuration at any management scope and associate the new key manager with a Secure Sockets Layer (SSL) configuration.

You must develop, package, and locate a Java Archive (.JAR) file for a custom key manager in the `was.install.root/lib/ext` directory on WebSphere Application Server. For more information, see rsec_ssldevcustomkeymgr.dita.

Complete the following steps in the administrative console:

1. Decide whether you want to create the custom key manager at the cell scope or below the cell scope at the node, server, or cluster, for example.

   Important: When you create a custom key manager at a level below the cell scope, you can associate it only with a Secure Sockets Layer (SSL) configuration at the same scope or higher. An SSL configuration at a scope lower than the key manager does not see the key manager configuration.

   - To create a custom key manager at the cell scope, click **Security > SSL certificate and key management > Key managers**. Every SSL configuration in the cell can select the key manager at the cell scope.
   - To create a custom key manager at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration* > Key managers**.
2. Click **New** to create a new key manager.
3. Type a unique key manager name.
4. Select the **Custom** implementation setting. With the custom setting, you can define a Java class that has an implementation on the Java interface javax.net.ssl.X509KeyManager and, optionally, the com.ibm.wsspi.ssl.KeyManagerExtendedInfo WebSphere Application Server interface. The standard implementation setting applies only when the key manager is already defined in the Java security provider list as a provider and an algorithm, which is not the case for a custom key manager. The typical standard key manager is `algorithm = IbmX509, provider = IBMJSSE2`.
5. Type a class name. For example, com.ibm.test.CustomKeyManager.
6. Select one of the following actions:
   - Click **Apply**, then click **Custom properties** under Additional Properties to add custom properties to the new custom key manager. When you are finished adding custom properties, click **OK** and **Save**, then go to the next step.
   - Click **OK** and **Save**, then go to the next step.
7. Click **SSL certificate and key management** in the page navigation at the top of the panel.
8. Select one of the following actions:
   - Click **SSL configurations** under Related Items for a cell-scoped SSL configuration.
   - Click **Manage endpoint security configurations** to select an SSL configuration at a lower scope.

9. Click the link for the existing SSL configuration that you want to associate with the new custom key manager. You can create a new SSL configuration instead of associating the custom key manager with an existing configuration. For more information, see “Creating a Secure Sockets Layer configuration” on page 1429.
10. Click **Trust and Key managers** under Additional Properties.
11. Select the new custom key manager in the **Key manager** drop-down list. If the new custom key manager is not listed, verify that you selected an SSL configuration scope that is at the same level or below the scope that you selected in Step 8.
12. Click **OK** and **Save**.

You have created a custom key manager configuration that references a JAR file in the installation directory of WebSphere Application Server and associates the custom configuration with an SSL configuration during the connection handshake.

You can create a custom key manager for a pure client. For more information, see “Commands for the keyManagerCommands group of the AdminTask object” on page 1679.

Example: Developing a custom key manager for custom Secure Sockets Layer key selection:

The following example is of a sample custom key manager. This simple key manager provides the alias if it is set, and defers to the custom key manager if it is not set. Additional code can be added, for example, to prompt for a key that is given to the issuers that are provided.

After you build and package a custom key manager, you can configure it from either the `ssl.client.props` file for a pure client or by using the SSLConfiguration KeyManager link in the administrative console. See “Key manager control of X.509 certificate identities” on page 1405 for more information about key managers.

Because only one key manager can be configured at a time for any given Secure Sockets Layer (SSL) configuration, the certificate selections on the server side might not work as they would when the default IbmX509 key manager is specified. When a custom key manager is configured, it is up to the owner of that key manager to ensure that the selection of the alias from the SSL configuration supplied is set properly when `chooseClientAlias` or `chooseServerAlias` are called. Look for the `com.ibm.ssl.keyStoreClientAlias` and `com.ibm.ssl.keyStoreServerAlias` SSL properties.

Note: This example should only be used as a sample, and is not supported.

```
package com.ibm.test;

import java.security.cert.X509Certificate;
import com.ibm.wsspi.ssl.KeyManagerExtendedInfo;

public final class CustomKeyManager
 implements javax.net.ssl.X509KeyManager, com.ibm.wsspi.ssl.KeyManagerExtendedInfo
{
    private java.util.Properties props = null;
    private java.security.KeyStore ks = null;
    private javax.net.ssl.X509KeyManager km = null;
    private java.util.Properties sslConfig = null;
    private String clientAlias = null;
    private String serverAlias = null;
    private int clientslotnum = 0;
    private int serverslotnum = 0;

    public CustomKeyManager()
    {
    }
```

```
/**
 * Method called by WebSphere Application Server runtime to set the custom
 * properties.
 *
 * @param java.util.Properties - custom props
 */
public void setCustomProperties(java.util.Properties customProps)
{
    props = customProps;
}

private java.util.Properties getCustomProperties()
{
    return props;
}

/**
 * Method called by WebSphere Application Server runtime to set the SSL
 * configuration properties being used for this connection.
 *
 * @param java.util.Properties - contains a property for the SSL configuration.
 */
public void setSSLConfig(java.util.Properties config)
{
    sslConfig = config;
}

private java.util.Properties getSSLConfig()
{
    return sslConfig;
}

/**
 * Method called by WebSphere Application Server runtime to set the default
 * X509KeyManager created by the IbmX509 KeyManagerFactory using the KeyStore
 * information present in this SSL configuration.  This allows some delegation
 * to the default IbmX509 KeyManager to occur.
 *
 * @param javax.net.ssl.KeyManager defaultX509KeyManager - default key manager for IbmX509
 */
public void setDefaultX509KeyManager(javax.net.ssl.X509KeyManager defaultX509KeyManager)
{
    km = defaultX509KeyManager;
}

public javax.net.ssl.X509KeyManager getDefaultX509KeyManager()
{
    return km;
}

/**
 * Method called by WebSphere Application Server runtime to set the SSL
 * KeyStore used for this connection.
 *
 * @param java.security.KeyStore - the KeyStore currently configured
 */
public void setKeyStore(java.security.KeyStore keyStore)
{
    ks = keyStore;
}
```

```
    public java.security.KeyStore getKeyStore()
    {
        return ks;
    }

/**
     * Method called by custom code to set the server alias.
     *
     * @param String - the server alias to use
     */
    public void setKeyStoreServerAlias(String alias)
    {
        serverAlias = alias;
    }

    private String getKeyStoreServerAlias()
    {
        return serverAlias;
    }

    /**
     * Method called by custom code to set the client alias.
     *
     * @param String - the client alias to use
     */
    public void setKeyStoreClientAlias(String alias)
    {
        clientAlias = alias;
    }

    private String getKeyStoreClientAlias()
    {
        return clientAlias;
    }

    /**
     * Method called by custom code to set the client alias and slot (if necessary).
     *
     * @param String - the client alias to use
     * @param int - the slot to use (for hardware)
     */
    public void setClientAlias(String alias, int slotnum) throws Exception
    {
        if ( !ks.containsAlias(alias))
        {
            throw new IllegalArgumentException ( "Client alias " + alias + "
            not found in keystore." );
        }
        this.clientAlias = alias;
        this.clientslotnum = slotnum;
    }

    /**
     * Method called by custom code to set the server alias and slot (if necessary).
     *
     * @param String - the server alias to use
     * @param int - the slot to use (for hardware)
     */
    public void setServerAlias(String alias, int slotnum) throws Exception
```

```
{
    if ( ! ks.containsAlias(alias))
    {
        throw new IllegalArgumentException ( "Server alias " + alias + "
        not found in keystore." );
    }
    this.serverAlias = alias;
    this.serverslotnum = slotnum;
}


/**
 * Method called by JSSE runtime to when an alias is needed for a client
 * connection where a client certificate is required.
 *
 * @param String keyType
 * @param Principal[] issuers
 * @param java.net.Socket socket (not always present)
 */
public String chooseClientAlias(String[] keyType, java.security.Principal[] issuers, java.net.Socket socket
{
    if (clientAlias != null && !clientAlias.equals(""))
    {
        String[] list = km.getClientAliases(keyType[0], issuers);
        String aliases = "";

        if (list != null)
        {
            boolean found=false;
            for (int i=0; i<list.length; i++)
            {
                aliases += list[i] + " ";
                if (clientAlias.equalsIgnoreCase(list[i]))
                    found=true;
            }

            if (found)
            {
                return clientAlias;
            }

        }
    }

    // client alias not found, let the default key manager choose.
    String[] keyArray = new String [] {keyType[0]};
    String alias = km.chooseClientAlias(keyArray, issuers, null);
    return alias.toLowerCase();
}

/**
 * Method called by JSSE runtime to when an alias is needed for a server
 * connection to provide the server identity.
 *
 * @param String[] keyType
 * @param Principal[] issuers
 * @param java.net.Socket socket (not always present)
 */
public String chooseServerAlias(String keyType, java.security.Principal[]
issuers, java.net.Socket socket)
```

```
    {
        if (serverAlias != null && !serverAlias.equals(""))
        {
            // get the list of aliases in the keystore from the default key manager
            String[] list = km.getServerAliases(keyType, issuers);
            String aliases = "";

            if (list != null)
            {
                boolean found=false;
                for (int i=0; i<list.length; i++)
                {
                    aliases += list[i] + " ";
                    if (serverAlias.equalsIgnoreCase(list[i]))
                        found = true;
                }

                if (found)
                {
                    return serverAlias;
                }
            }
        }

        // specified alias not found, let the default key manager choose.
        String alias = km.chooseServerAlias(keyType, issuers, null);
        return alias.toLowerCase();
    }

    public String[] getClientAliases(String keyType, java.security.Principal[] issuers)
    {
        return km.getClientAliases(keyType, issuers);
    }

    public String[] getServerAliases(String keyType, java.security.Principal[] issuers)
    {
        return km.getServerAliases(keyType, issuers);
    }

    public java.security.PrivateKey getPrivateKey(String s)
    {
        return km.getPrivateKey(s);
    }

    public java.security.cert.X509Certificate[] getCertificateChain(String s)
    {
        return km.getCertificateChain(s);
    }

    public javax.net.ssl.X509KeyManager getX509KeyManager()
    {
        return km;
    }

}
```

Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint

After you create a Secure Sockets Layer (SSL) configuration, you must associate a secure outbound management scope with the new configuration. In this release, you can associate one SSL configuration

with one remote secure endpoint and a different SSL configuration to another remote secure endpoint. Both endpoints can use the same outbound protocol, if appropriate. This task describes how to create the association dynamically.

Dynamic outbound selection requires that you provide only the outbound protocol name, the target host, and the target port so that WebSphere Application Server can make a connection between the SSL configuration and the outbound protocol or remote secure endpoint. The dynamic outbound selection method takes precedence over other selection methods, such as central management and direct selection, but is second to the programmatic method, that is, setting an SSL configuration on the running thread. For more information about the selection types and precedence rules, see “Secure communications using Secure Sockets Layer” on page 1394.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Outbound**.
2. Select the management scope that you want to associate with an SSL configuration on the topology tree.
3. Under Related Items, click **Dynamic outbound endpoint SSL configurations**. The default dynamic outbound configuration name, the target protocol, host, and port connection information, and the SSL configuration name display.
4. Click **New** to create a new dynamic outbound configuration.
5. Type a dynamic outbound configuration name. Use a name that is descriptive of the purpose of the dynamic selection configuration.
6. Optionally, type a dynamic selection configuration description.
7. Type the connection information that you want to associate with the configuration that is displayed in the SSL configuration drop-down list. The connection information must be in the format *protocol name*, *target host*, *target port*. You can substitute an asterisk (*) for any value, as in the following examples:
   - *,*,443
   - *,www.ibm.com,443
   - HTTP,.austin.ibm.com,*

   where 443 is a port, www.ibm.com is a host, HTTP is a protocol, and .austin.ibm.com is a target host. You can add multiple connections, but each additional connection can affect outbound performance.
8. Click **Add** to add the new connection to the set of SSL configuration connections. To remove a connection, select it and click **Remove**.
9. Select an SSL configuration from the list.
10. Click **Get certificate aliases** to refresh the certificate aliases that are contained in the associated key store.
11. Choose a certificate alias from the list.
12. Click **OK** and **Save**.

WebSphere Application Server is ready to connect one or more SSL configurations to one or more remote secure endpoints.

You can return to the outbound tree and select another management scope to associate with the same or a new outbound configuration.

Example: Programmatically specifying an outbound SSL configuration using JSSEHelper API:

WebSphere Application Server provides a way to specify programmatically which Secure Sockets Layer (SSL) configurations to use prior to making an outbound connection. The com.ibm.websphere.ssl.JSSEHelper interface provides a complete set of application programming interfaces (APIs) for handling SSL configurations.

You must perform the following steps for your application when using the JSSEHelper API to establish an SSL properties object on the thread for use by the runtime. Some of these APIs have Java 2 Security permission requirements. See the JSSEHelper API documentation for more information about the permissions required by your application.

1. Obtain an instance of the JSSEHelper API by typing the following command:

   ```
   com.ibm.websphere.ssl.JSSEHelper jsseHelper
   = com.ibm.websphere.ssl.JSSEHelper.getInstance();
   ```

2. Obtain SSL properties from the WebSphere Application Server configuration or use those provided by your application. You can obtain these properties in several ways:
   - By direction selection of an alias name, within the same management scope or higher as in the following example:

     ```
     try
     {
      String alias = "NodeAServer1SSLSettings";
     // As specified in the WebSphere SSL configuration
      Properties sslProps = jsseHelper.getProperties(alias);
     }
     catch (com.ibm.websphere.ssl.SSLException e)
     {
      e.printStackTrace();   // handle exception
     }
     ```

   - By using the getProperties API for programmatic, direction, dynamic outbound, or management scope selection (based on precedence rules and inheritance). The SSL runtime uses the getProperties API to determine which SSL configuration to use for a particular protocol. This decision is based on both the input (sslAlias and connectionInfo) and the management scope from which the property is called. The getProperties API makes decisions in the following order:
     a. The API checks the thread to see if properties already exist.
     b. The API checks for a dynamic outbound configuration that matches the ENDPOINT_NAME, REMOTE_HOST, and or REMOTE_PORT.
     c. The API checks to see if the optional sslAlias property is specified. You can configure any protocol as direct or centrally managed. When a protocol is configured as direct, the sslAlias parameter is `null`. When a protocol is configured as centrally managed, the sslAlias parameter is also `null`.
     d. If no selection has been made, the API chooses the dynamic outbound configuration based on the management scope it was called from. If the dynamic outbound configuration is not defined in the same scope, it then searches the hierarchy to locate one.

     The last choice is the cell-scoped SSL configuration (in Network Deployment) or the node-scoped SSL configuration (in Base Application Server). The com.ibm.websphere.ssl.SSLConfigChangeListener parameter is notified when the SSL configuration that is chosen by a call to the getProperties API changes. The protocol can then call the API again to obtain the new properties as in the following example:

     ```
     try
     {
      String sslAlias = null;  // The sslAlias is not specified directly at this time.
      String host = "myhost.austin.ibm.com";  // the target host
      String port = "443";  // the target port

      HashMap connectionInfo = new HashMap();
      connectionInfo.put(JSSEHelper.CONNECTION_INFO_DIRECTION,
       JSSEHelper.DIRECTION_OUTBOUND);
      connectionInfo.put(JSSEHelper.CONNECTION_INFO_REMOTE_HOST, host);
      connectionInfo.put(JSSEHelper.CONNECTION_INFO_REMOTE_PORT,
     ```

```
 Integer.toString(port));
 connectionInfo.put(JSSEHelper.CONNECTION_INFO_ENDPOINT_NAME,
  JSSEHelper.ENDPOINT_IIOP);

 java.util.Properties props = jsseHelper.getProperties(sslAlias,
  connectionInfo, null);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
 e.printStackTrace();   // handle exception
}
```

- By creating your own SSL properties and then passing them to the runtime, as in the following example:

```
try
{
 // This is the recommended "minimum" set of SSL properties. The trustStore can
 // be the same as the keyStore.
 Properties sslProps = new Properties();
 sslProps.setProperty("com.ibm.ssl.trustStore", "some value");
 sslProps.setProperty("com.ibm.ssl.trustStorePassword", "some value");
 sslProps.setProperty("com.ibm.ssl.trustStoreType", "some value");
 sslProps.setProperty("com.ibm.ssl.keyStore", "some value");
 sslProps.setProperty("com.ibm.ssl.keyStorePassword", "some value");
 sslProps.setProperty("com.ibm.ssl.keyStoreType", "some value");

 jsseHelper.setSSLPropertiesOnThread(sslProps);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
 e.printStackTrace();   // handle exception
}
```

3. Use the JSSEHelper.setSSLPropertiesOnThread(props) API to set the Properties object on the thread so that the runtime picks it up and uses the same JSSEHelper.getProperties API. You can also obtain properties from the thread after they are set with the jsseHelper.getSSLPropertiesOnThread() API, as in the following example:

```
try
{
 Properties sslProps = jsseHelper.getProperties(null,
  connectionInfo, null); jsseHelper.setSSLPropertiesOnThread(sslProps);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
 e.printStackTrace();   // handle exception
}
```

4. When the connection is completed, you must clear the SSL properties from the thread by passing the `null` value to the setPropertiesOnThread API, as in the following example:

```
try
{
 jsseHelper.setSSLPropertiesOnThread(null);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
 e.printStackTrace();   // handle exception
}
```

Select the approach that best fits your connection situation when you specify programmatically which Secure Sockets Layer (SSL) configurations to use prior to making an outbound connection.

Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes:

After you create a Secure Sockets Layer (SSL) configuration, you must associate a secure inbound or outbound management scope with the new configuration. You can manage the association centrally so

that you can easily make changes that affect all the scopes that are lower on the topology and that are associated with the configuration. Beginning with WebSphere Application Server version 6.1, the recommended and the default configuration method is centrally managed SSL configurations.

You can simplify the number of associations that you need to make for an SSL configuration by associating the configuration with the highest level management scope requiring a unique configuration. SSL configuration associations manifest inheritance behaviors. Because of the inheritance behaviors, all of the scopes that are lower on the topology inherit this SSL configuration. For example, an association you make at the cell level affects nodes, servers, clusters, and endpoints. For more information, see “Central management of Secure Sockets Layer configurations” on page 1410.

A precedence rule determines which SSL configuration association is used at a particular scope. The highest precedence is given to endpoints on the topology. If you establish an association at the endpoint, this association overrides any prior association that you made higher up on the management scope topology.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management**.
2. Select the **Dynamically update the runtime when SSL configuration changes** check box if you want changes that you make to an existing SSL configuration to occur dynamically. All outbound SSL communications honor the dynamic SSL changes. Protocols that do not use the channel frameworks SSL channel for inbound communications, including Object Request Broker (ORB) and administrative SOAP protocols, do not honor dynamic updates. For more information, see “Dynamic configuration updates” on page 1424.
3. Click **Manage endpoint security configurations**.
4. Select either the inbound or the outbound tree. After finishing the selected tree, you can return to this step to repeat the following steps for the other tree.
5. Click the link for the selected cell, node, node group, server, cluster, or endpoint on the topology tree. If the scope already has an associated SSL configuration and alias, these objects display in parentheses immediately following the scope name, for example: Node01(NodeDefaultSSLSettings,default). If the deployment manager has federated a node, the node scope SSL configuration overrides the cell scope configuration above it in the topology.
6. Decide whether to override the inherited values that display in the read-only fields. Read-only fields include the management scope name, the direction, and the inherited SSL configuration name and certificate alias.
   - If you are satisfied with these values, do not override them.
   - If you want to override the inherited values, select the **Override inherited values** check box.
7. Select an SSL configuration from the list.
8. Click **Update certificate alias list**. The certificate alias list comes from the key store that is referenced by the new SSL configuration.
9. Click **Manage certificates** if you want to manage the personal certificates that are contained in the key store that is referenced in the SSL configuration.
10. Click **Update certificate alias list** to refresh the list of aliases.
11. Select a certificate alias in the key store to represent the identity of the endpoint.
12. Click **OK** to save your changes.
13. Click **Manage endpoint security configurations and trust zones** to return to the topology tree.
14. Configure the opposite direction on the topology tree using the steps in this task. You can also select additional scopes to associate with the SSL configuration, as needed.

Each SSL configuration at the selected scope and at scopes beneath it on the topology tree have the same SSL configuration properties. The following SSL configuration methods override the centrally managed configurations that you associate in the tree view:

- Direct selection at the endpoint
- Dynamic outbound SSL configuration associations
- Programmatic specifications

At any management scope, you can configure the following objects: dynamic outbound endpoint SSL configurations, key stores, key sets, key set groups, key managers, and trust managers. Like SSL configurations, these objects are scoped automatically so that they are not visible higher up in the tree nor are they loaded during runtime by processes that are higher up in the tree.

Selecting an SSL configuration alias directly from an endpoint configuration:

You can associate a secure outbound endpoint with a new Secure Sockets Layer (SSL) configuration directly. If you are migrating from a release prior to version 6.1, WebSphere Application Server still supports configurations that were selected directly at an endpoint. Direct selection always overrides centrally managed configurations and preserves migrated configurations.

Select an SSL configuration alias directly at the following endpoints:

- **Security > Secure administration, applications, and infrastructure > RMI/IIOP security > CSIv2 outbound transport**
- **Security > Secure administration, applications, and infrastructure > RMI/IIOP security > CSIv2 inbound transport**
- **System administration > Deployment manager > Transport Chain > WCInboundAdminSecure > SSL inbound channel (SSL_1)**
- **System administration > Deployment manager > Administration Services > JMX connectors > SOAPConnector > Custom Properties > sslConfig**
- **System administration > Node agents > nodeagent > Administration Services > JMX connectors > SOAPConnector > Custom Properties > sslConfig**
- **Servers > Application servers > server1 > Messaging engine inbound transports > InboundSecureMessaging > SSL inbound channel (SIB_SSL_JFAP)**
- **Servers > Application servers > server1 > WebSphere MQ link inbound transports > InboundSecureMQLink > SSL inbound channel (SIB_SSL_MQFAP)**
- **Servers > Application servers > server1 > SIP Container Settings > SIP container transport chains > SIPCInboundDefaultSecure > SSL inbound channel (SSL_5)**
- **Servers > Application servers > server1 > Web Container Settings > Web container transport chains > WCInboundAdminSecure > SSL inbound channel (SSL_1)**
- **Servers > Application servers > server1 > Web Container Settings > Web container transport chains > WCInboundDefaultSecure > SSL inbound channel (SSL_2)**

Attention: Keep in mind that central management of SSL configurations can be a more efficient strategy because multiple configurations can be contained within a single SSLConfigGroup. If you need to convert configuration references that are already directly managed to centrally managed configurations, modify each endpoint individually. For more information on specific wsadmin commands, see “Commands for the SSLConfigGroupCommands group of the AdminTask object” on page 1683.

Complete the following steps in the administrative console:

Note: These steps provide an example to follow when you directly select any of the endpoints listed above.

1. Click **Security > Secure administration, applications, and infrastructure > RMI/IIOP security > CSIv2 outbound transport**.
2. Click **Use specific SSL alias**. When you identify a specific SSL alias, you override the centrally managed scope associations.

3. Select an SSL configuration alias from the drop-down list.
4. Click **OK**.
5. Repeat these steps for additional protocols or endpoints, if desired.

By associating the endpoint directly, you have overridden a centrally managed SSL configuration.

If you decide to use management scopes instead of endpoints to associate an SSL configuration, follow the steps above, but click **Centrally managed** instead of **Use specific SSL alias**, then click **Manage endpoint security configurations**. The console is redirected to **Security > SSL certificate and key management > Manage endpoint security configurations**.

Enabling Secure Sockets Layer client authentication for a specific inbound endpoint:

When you establish a Secure Sockets Layer (SSL) configuration, you can enable client authentication for a specific inbound endpoint.

The endpoint configuration must already exist in the SSL topology.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound >** *SSL_configuration*. If you want to enable SSL client authentication for all processes, define an SSL configuration for the new endpoint at the node or cell level so that it is visible to all processes on the same node or on the entire cell. For more information, see "Creating a Secure Sockets Layer configuration" on page 1429.
2. Select **Override inherited values**. The SSL configuration is used for the current scope and any lower scopes that have not already designated an SSL configuration. This field displays for server and node groups within the object hierarchy and does not display for the top-level node or cell.
3. Select an SSL configuration from the drop-down list.
4. Click **Update certificate alias list**.
5. Select a **Certificate alias** from the drop-down list.
6. Click **OK** to save the configuration.

You can repeat the previous steps for each endpoint that uses the same SSL configuration to enable client authentication for the inbound endpoints.

CSIv2 Protocol Exception:

The Common Secure Interoperability Version 2 (CSIv2) secure endpoints, used for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security, cannot override inherited values. While the rest of the SSL properties are effective for CSIv2 when they are selected at the centrally-managed Secure Communications panel, the client authentication selection is controlled by the CSIv2 protocol configuration.

To enable SSL client certificate authentication for the CSIv2 protocol, you must use the CSIv2 inbound and outbound authentication panels. For SSL client authentication to occur between two servers, you must enable (support or require) SSL client certificate authentication for both the inbound and the outbound policies.

WebSphere Application Server can either request (support) clients to provide signer certificates for the SSL handshake, or the server can require clients to provide a valid signer certificates for the SSL handshake, which is a more secure method. However, when the server requires certificates, the server must obtain a signer for each client that connects to the server, which involves more server-side management.

The client certificate should not be used for the identity when it is used from server-to-server. However, when a pure client sends the client certificate it is used for the identity unless a message level identity is specified, such as a user ID or a password.

Do the following to enable client certificate authentication for the CSIv2 protocol for server-to-server:

1. **Click Security > Secure administration, applications, and infrastructure**.
2. Expand the **RMI/IIOP** security section.
3. Click **CSIv2 inbound authentication**.
4. Under Client authentication, select either **supported** or **required**. When you select required, only one SSL port is opened (CSV2_SSL_MUTUALAUTH_LISTENER_ADDRESS). When you select supported, two SSL ports are opened (both CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS and CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS).

   If there are two ports, the client can select either based on the security configuration policy of the port.
5. Click **OK** to save.
6. If you want server-to-server SSL client authentication, then complete the remaining steps. If you don't complete the remaining steps, only pure clients are enabled to send client certificates.
7. Expand the **RMI/IIOP** security section.
8. Click **CSIv2 outbound authentication**.
9. Under Client authentication, select either **supported** or **required**.

The SSL configuration for the inbound secure endpoints for which you enable SSL client certificate authentication must have the signer certificate from any client that attempts to open a connection to that inbound secure endpoint. You must collect those signers and then add them to the trust store associated with the inbound secure endpoints SSL configuration.

Manage endpoint security configurations:

Use this page to select a Secure Socket Layer (SSL) configuration from the Local Topology hierarchy, which includes cells, nodes, node groups, servers, and clusters.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations**.

Local topology:

The Local topology represents the hierarchy of nodes, node groups, clusters, servers, and end points within the cell that comprise a centralized SSL configuration.

The topology acts as a hierarchical tree in terms of inheritance. For example, if an SSL configuration has been associated with a specific node, then all servers within that node will inherit that SSL configuration selection, provided the servers are not associated with an SSL configuration at the server scope. Centralized management of SSL is the default configuration; however, it can be overridden at various locations to directly select a specific SSL alias as in previous releases for backwards compatibility.

Scope	Description
Inbound/Outbound	Specifies the topology tree in terms of connection direction. For example, the inbound tree represents all server endpoints that receive connections at the various servers within the cell. The outbound tree represents the client side of connections from the various servers within the cell.

Scope	Description
Nodes	Specifies the nodes that are part of the cell. The list of nodes is updated anytime a node gets federated into the cell.
Servers	Specifies the servers that are part of a specific node. You can enable a specific server to have an SSL configuration associated with it so that resources within the same server can use the associated SSL configuration.
Clusters	Specifies the clusters that are part of the cell. When an SSL configuration is associated with a cluster, all servers within the cluster will use the same SSL configuration unless specified at a lower level in the topology.
Nodegroups	Specifies the node groups that are part of the cell. When an SSL configuration is associated with a node group, all nodes within that node group may use the same SSL configuration unless one is specified at a lower scope in the topology or the specific end point has chosen a direct alias reference.
Secure port and transport	Specifies an endpoint name to associate with an SSL configuration when more specific SSL settings are needed at this level. You could select an alias directly at the endpoint panel; however, when you use **Secure port and transport**, you can maintain more centralized control of the SSL configuration and make changes more easily.

Dynamic inbound and outbound endpoint SSL configurations collection:

Use this page to manage dynamic endpoint Secure Sockets Layer (SSL) configurations, which represent associations between Secure Socket Layer (SSL) configurations and their target protocol, host, and port.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Dynamic [inbound | outbound] endpoint SSL configurations**.

When an outbound connection is attempted, this association is checked ahead of the SSL configuration scope association. Based on the target protocol,host,port, the outbound SSL configuration used can be different from the default specified in the SSL scope configuration.

Button	Resulting action
New	Adds a new dynamic outbound selection criteria. The outbound connection selects an SSL configuration based upon connection information, including DNS host name and domain, port, and protocol type. When an outbound connection is being made, the dynamic outbound selection criteria are queried for a match, and if found the SSL configuration associated is used.
Delete	Deletes an existing dynamic outbound endpoint SSL configuration.

Name:

Specifies the unique name of the dynamic endpoint configuration.

Connection information:

Specifies the set of target protocol, host, port for the outbound request in the form *protocol*,*host*,*port*.

SSL Configuration:

Specifies the SSL configuration that is used by requests at this scope when a match occurs for the given selection criteria.

Dynamic outbound endpoint SSL configuration settings:

Use this page to set properties for dynamic outbound endpoint SSL configurations, which represent associations between SSL configurations and their target protocol, host, and port.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Dynamic [inbound | outbound] endpoint SSL configurations > New**.

When an outbound connection is attempted, this association is checked ahead of the Secure Sockets Layer (SSL) configuration scope association. This means based on the target protocol,host,port, the outbound SSL configuration used can be different than the default specified in the SSL scope configuration.

Name:

Specifies the unique name of the dynamic endpoint configuration.

Data type: Text

Description:

Specifies text that describes the purpose of this dynamic selection criteria.

Data type: Text

Add connection information:

Specifies select information in the form protocol,host,port for the outbound connection. Multiple selection criteria can be entered. An asterisk (*) can be used to mean all protocols, hosts, or ports. You can use an * for any field.

Data type: Text

An example of selection criteria is `*,www.ibm.com,*`, which means that any time the target host is www.ibm.com, you must use the SSL configuration specified here. Another example selection criteria is `IIOP,*,*`, which means that any outbound IIOP request uses the SSL configuration that is specified in the SSL configuration field. When there is a conflict between two selection criteria, the application server uses the first match. The list of valid protocols you can use include: IIOP, HTTP, JMS, LDAP, SIP, ADMIN_SOAP, or ADMIN_IIOP.

Add:

Specifies to add the selected information from the **Add select information** menu to the right-hand list.

Remove:

Specifies to remove the selection from the right-hand list.

SSL Configuration:

Specifies the SSL configuration to be used by requests at this scope when a match occurs for the given selection criteria.

Data type: Text

Get certificate alias:

When selected, the keystore within the selected SSL configuration is queried for a list of personal certificates from which to choose.

Certificate alias:

Specifies the certificate alias that is used as the identity for the connection.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the keystore, the key manager might not consistently select the same certificate.

Data type: Text
Default: (none)

Quality of protection (QoP) settings

Use this page to specify security level, ciphers, and mutual authentication settings for the Secure Socket Layer (SSL) configuration.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations >** {**Inbound | Outbound** *SSL_configuration_name*}. Under **Related Items**, click **SSL configurations >** {*SSL_configuration_name* | **New**}. Under **Additional Properties**, click **Quality of protection (QoP) settings**.

Client authentication:

Specifies the whether SSL client authentication should be requested if the SSL connection is used for the server side of the connection.

If `None` is selected, the server does not request that a client certificate be sent during the handshake. If `Supported` is selected, the server requests that a client certificate be sent. If the client does not have a certificate, the handshake might still succeed. If `Required` is selected, the server requests that a client certificate be sent. If the client does not have a certificate, the handshake fails.

Data type: Text
Default: None

Protocol:

Specifies the Secure Sockets Layer (SSL) handshake protocol. This protocol is typically SSL_TLS, which supports all handshake protocols except for SSLv2 on the server side. When United States Federal Information Processing standard (FIPS) option is enabled, Transport Layer Security (TLS) is automatically used regardless of this setting.

Data type: text
Default: SSL_TLS

Predefined JSSE provider:

Specifies one of the predefined Java Secure Sockets Extension (JSSE) providers. The IBMJSSE2 provider is recommended for use on all platforms which support it. It is required for use by the channel framework SSL channel. When Federal Information Processing Standard (FIPS) is enabled, IBMJSSE2 is used in combination with the IBMJCEFIPS crypto provider.

Default: Enabled

Select provider:

Specifies a package that implements a subset of the cryptography aspects for the Java security application programming interface (API). This value is a JSSE provider name that is listed in the java.security file. Note that cipher suites and protocol values depend upon the provider.

Data type: Text
Default: IBMJSSE2

Custom JSSE provider:

Specifies that a custom JSSE provider should be used.

Default: Disabled

Custom provider:

Specifies a package that implements a subset of the cryptography aspects for the Java security application programming interface (API). This value is a Java Secure Sockets Extension (JSSE) provider name that is listed in the java.security file. Note that cipher suites and protocol values depend upon the provider.

Data type: Text

Cipher suite groups:

Specifies the various cipher suite groups that can be chosen depending upon your security needs. The stronger the cipher suite strength, the better the security; however, this can result in performance consequences.

Data type: Text
Default: Strong

Update selected ciphers:

When selected, the cipher suites that are contained within the selected Cipher suite group are added to the list of **Selected ciphers**. Any change to this list changes the Cipher suite group to `custom`.

Selected ciphers:

Specifies the ciphers that are effective when the configuration is saved. These ciphers are used to negotiate with the remote side of the connection during the handshake. A common cipher needs to be selected or the handshake fails.

Data type: Text

Add:

Specifies to add the selected cipher to the **Selected ciphers** list.

Remove:

Specifies to remove the selected cipher from the **Selected ciphers** list.

ssl.client.props client configuration file

Use the `ssl.client.props` file to configure Secure Sockets Layer (SSL) for clients. In previous releases of WebSphere Application Server, SSL properties were specified in the `sas.client.props` or `soap.client.props` files or as system properties. By consolidating the configurations, WebSphere Application Server enables you to manage security in a manner that is comparable to server-side configuration management. You can configure the `ssl.client.props` file with multiple SSL configurations.

Setting up the SSL configuration for clients

Client runtimes are dependent on the WebSphere Application Server `ssl.client.props` configurations.

Use the `setupCmdLine.bat` or `setupCmdLine.sh` script on the command line to specify the com.ibm.SSL.ConfigURL system property.

```
-Dcom.ibm.SSL.ConfigURL=file:C:\WebSphere\AppServer\profiles\default
\properties\ssl.client.props
```

The com.ibm.SSL.ConfigURL property references a file URL that points to the `ssl.client.props` file. You can reference the CLIENTSSL variable on the command line of any script that uses the `setupCmdLine.bat` or `setupCmdLine.sh` file.

When you specify the com.ibm.SSL.ConfigURL system property, the SSL configuration is available to all protocols that use SSL. SSL configurations, which are referenced in the `ssl.client.props` file, also have aliases that you can reference. In the following sample code from the `sas.client.props` file, all of the SSL properties are replaced with a property that points to an SSL configuration in the `ssl.client.props` file:

```
com.ibm.ssl.alias=default
```

The following sample code shows a property in the `soap.client.props` file that is similar to the com.ibm.SSL.ConfigURL property. This property references a different SSL configuration on the client side:

```
com.ibm.ssl.alias=ADMIN_SOAP
```

In the `ssl.client.props` file, you can change the administrative SSL configuration to avoid modifying the `soap.client.props` file.

Tip: In WebSphere Application Server Version 6.1, support for SSL properties is still specified in the `sas.client.props` and `soap.client.props` files. However, consider moving the SSL configurations to the `ssl.client.props` file, because this file is the new configuration model for client SSL.

Properties of the ssl.client.props file

This section describes the default `ssl.client.props` file properties in detail, by sections within the file.

Global properties

Global SSL properties are process-specific properties that include Federal Information Processing Standard (FIPS) enablement, the default SSL alias, the user.root property for specifying the root location of the key and truststore paths, and so on.

Property	Default	Description
com.ibm.ssl.defaultAlias	DefaultSSLSettings	Specifies the default alias that is used whenever an alias is not specified by the protocol that calls the JSSEHelper API to retrieve an SSL configuration. This property is the final arbiter on the client side for determining which SSL configuration to use.
com.ibm.ssl.validationEnabled	false	When set to `true`, this property validates each SSL configuration as it is loaded. Use this property for debug purposes only, to avoid unnecessary performance overhead during production.
com.ibm.ssl.performURLHostNameVerification	false	When set to `true`, this property enforces URL host name verification. When HTTP URL connections are made to target servers, the common name (CN) from the server certificate must match the target host name. Without a match, the host name verifier rejects the connection. The default value of `false` omits this check. As a global property, it sets the default host name verifier. Any javax.net.ssl.HttpsURLConnection object can choose to enable host name verification for that specific instance by calling the setHostnameVerifier method with its own HostnameVerifier instance.
com.ibm.security.useFIPS	false	When set to `true`, FIPS-compliant algorithms are used for SSL and other Java Cryptography Extension (JCE)-specific applications. This property is typically not enabled unless the property is required by the operating environment.
user.root	C:\WebSphere\ AppServer \profiles\default	This property can be used by key and truststore location properties as a single property for specifying the root path to the key and truststores. Typically, this property is the profile root. However, you can modify this property to any root directory on the local machine that has the proper read and potentially write authority to that directory.
profile_root of the profile	C\WebSphere\ AppServer \profiles\default	Key and truststore location properties can use this property as a single property for specifying the root path to the key and truststores. You can modify this property to any root directory on the local machine that has the proper read (and potentially write) authority to that directory.

Certificate creation properties

Use certificate creation properties to specify the default self-signed certificate values for the major attributes of a certificate. You can define the distinguished name (DN), expiration date, key size, and alias that are stored in the keystore.

Property	Default	Description
com.ibm.ssl.defaultCertReqAlias	default_alias	This property specifies the default alias to use to reference the self-signed certificate that is created in the keystore. If the alias already exists with that name, the default alias is appended with `_#`, where the number sign (#) is an integer that starts with 1 and increments until it finds a unique alias.

Property	Default	Description
com.ibm.ssl.defaultCertReqSubjectDN	cn=${*hostname*}, o=IBM,c=US	This property uses the property distinguished name (DN) that you set for the certificate when it is created. The ${*hostname*} variable is expanded to the host name on which it resides. You can use correctly formed DNs as specified by the X.509 certificate.
com.ibm.ssl.defaultCertReqDays	365	This property specifies the validity period for the certificate and can be as small as 1 day and as large as the maximum number of days that a certificate can be set, which is approximately 20 years.
com.ibm.ssl.defaultCertReqKeySize	1024	This property is the default key size. The valid values depend upon the Java Virtual Machine (JVM) security policy files that are installed. By default, the product JVMs ship with the export policy file that limits the key size to 1024. To get a large key size such as 2048, you can download the restricted policy files from the Web site.

SSL configuration properties

Use the SSL configuration properties section to set multiple SSL configurations. For a new SSL configuration specification, set the com.ibm.ssl.alias property because the parser starts a new SSL configuration with this alias name. The SSL configuration is referenced by using the alias property from another file, such as `sas.client.props` or `soap.client.props`, through the default alias property. The properties that are specified in the following table enable you to create a javax.net.ssl.SSLContext, among other SSL objects.

Property	Default	Description
com.ibm.ssl.alias	DefaultSSLSettings	This property is the name of this SSL configuration and must be the first property for an SSL configuration because it references the SSL configuration. If you change the name of this property after it is referenced elsewhere in the configuration, the runtime defaults to the com.ibm.ssl.defaultAlias property whenever the reference is not found. The error `trust file is null` or `key file is null` might display when you start an application using an SSL reference that is no longer valid.
com.ibm.ssl.protocol	SSL_TLS	This property is the SSL handshake protocol that is used for this SSL configuration. This property attempts Transport Layer Security (TLS) first, but accepts any remote handshake protocol, including SSLv3 and TLS. Valid values for this property include SSLv2 (client side only), SSLv3, SSL, TLS, TLSv1, and SSL_TLS.
com.ibm.ssl.securityLevel	HIGH	This property specifies the cipher group that is used for the SSL handshake. The typical selection is HIGH, which specifies 128-bit or higher ciphers. The MEDIUM selection provides 40-bit ciphers. The LOW selection provides ciphers that do not perform encryption, but do perform signing for data integrity. If you specify your own cipher list selection, uncomment the property com.ibm.ssl.enabledCipherSuites.

Property	Default	Description
com.ibm.ssl.trustManager	IbmX509	This property specifies the default trust manager that you must use to validate the certificate sent by the target server. This trust manager does not perform certificate revocation list (CRL) checking. You can choose to change this to IbmPKIX for CRL checking using CRL distribution lists in the certificate, which is a standard way to perform CRL checking. When you want to perform custom CRL checking, you must implement a custom trust manager and specify the trust manager in the com.ibm.ssl.customTrustManagers property. The IbmPKIX option might affect performance because this option requires IBMCertPath for trust validation. Use IbmX509 unless CRL checking is necessary.
com.ibm.ssl.keyManager	IbmX509	This property specifies the default key manager to use for choosing the client alias from the specified keystore. This key manager uses the com.ibm.ssl.keyStoreClientAlias property to specify the keystore alias. If this property is not specified, the choice is delegated to Java Secure Socket Extension (JSSE). JSSE typically chooses the first alias that it finds.
com.ibm.ssl.contextProvider	IBMJSSE2	This property is used to choose the JSSE provider for the SSL context creation. It is recommended that you default to IBMJSSE2 when you use a Java virtual machine (JVM). The client plug-in can use the SunJSSE provider when using a Sun JVM.
com.ibm.ssl.enableSignerExchangePrompt	true	This property determines whether to display the signer exchange prompt when a signer is not present in the client truststore. The prompt displays information about the remote certificate so that WebSphere Application Server can decide whether or not to trust the signer. It is very important to validate the certificate signature (hash). This signature is the only reliable information that can guarantee that the certificate has not been modified from the original server certificate. For automated scenarios, disable this property to avoid SSL handshake exceptions. Run the `retrieveSigners.bat` file or the `retrieveSigners.sh` script, which sets up the SSL signer exchange, to download the signers from the server prior to running the client.
com.ibm.ssl.keyStoreClientAlias	default	This property is used to reference an alias from the specified keystore when the target does not request client authentication. When WebSphere Application Server creates a self-signed certificate for the SSL configuration, this property determines the alias and overrides the global com.ibm.ssl.defaultCertReqAlias property.

Property	Default	Description
com.ibm.ssl.customTrustManagers	Commented out by default	This property enables you to specify one or more custom trust managers, which are separated by commas. These trust managers can be in the form of *algorithm*|*provider* or *classname*. For example, IbmX509IIBMJSSE2 is in the*algorithm*|*provider* format, and the com.acme.myCustomTrustManager interface is in the *classname* format. The class must implement the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface. These trust managers run in addition to the default trust manager that is specified by the com.ibm.ssl.trustManager interface. These trust managers do not replace the default trust manager.
com.ibm.ssl.customKeyManager	Commented out by default	This property enables you to have one, and only one, custom key manager. The key manager replaces the default key manager that is specified in the com.ibm.ssl.keyManager property. The form of the key manager is *algorithm*|*provider* or *classname*. See the format examples for the com.ibm.ssl.customTrustManagers property. The class must implement the javax.net.ssl.X509KeyManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface. This key manager is responsible for alias selection.
com.ibm.ssl.dynamicSelectionInfo	Commented out by default	This property enables dynamic association with the SSL configuration. The syntax for a dynamic association is *outbound_protocol*, *target_host*, or *target_port*. For multiple specifications, use the vertical bar (|) as the delimiter. You can replace any of these values with an asterisk (*) to indicate a wildcard value. Valid *outbound_protocol* values include: IIOP, HTTP, LDAP, SIP, BUS_CLIENT, BUS_TO_WEBSPHERE_MQ, BUS_TO_BUS, and ADMIN_SOAP. When you want the dynamic selection criteria to choose the SSL configuration, uncomment the default property, and add the connection information. For example, add the following on one line `com.ibm.ssl.dynamicSelectionInfo=HTTP, .ibm.com,443|HTTP,.ibm.com,9443` . ADMIN_IIOP, and ADMIN_SOAP represent administrative uses of these protocols for the Remote Method Invocation (RMI) and SOAP connectors, respectively. In the SSL configuration alias AdminSOAPSSLSettings, you can set the value to `ADMIN_SOAP,*,*`, which indicates that any outbound administrative SOAP connection can use this SSL configuration. Dynamic selection takes precedence over direct alias selection in the `soap.client.props` file.
com.ibm.ssl.enabledCipherSuites	Commented out by default	This property enables you to specify a custom cipher suite list and override the group selection in the com.ibm.ssl.securityLevel property. The valid list of ciphers varies according to the provider and JVM policy files that are applied. For cipher suites, use a space as the delimiter.

Property	Default	Description
com.ibm.ssl.keyStoreName	ClientDefaultKeyStore	This property references a keystore configuration name. If you have not already defined the keystore, the rest of the keystore properties must follow this property. After you define the keystore, you can specify this property to reference the previously specified keystore configuration. New keystore configurations in the `ssl.client.props` file have a unique name.
com.ibm.ssl.trustStoreName	ClientDefaultTrustStore	This property references a truststore configuration name. If you have not already defined the truststore, the rest of the truststore properties must follow this property. After you define the truststore, you can specify this property to reference the previously specified truststore configuration. New truststore configurations in the `ssl.client.props` file should have a unique name.

Keystore configurations

SSL configurations reference keystore configurations whose purpose is to identify the location of certificates. Certificates represent the identity of clients that use the SSL configuration. You can specify keystore configurations with other SSL configuration properties. However, it is recommended that you specify the keystore configurations in this section of the `ssl.client.props` file after the com.ibm.ssl.keyStoreName property identifies the start of a new keystore configuration. After you fully define the keystore configuration, the com.ibm.ssl.keyStoreName property can reference the keystore configuration at any other point in the file.

Property	Default	Description
com.ibm.ssl.keyStoreName	ClientDefaultKeyStore	This property specifies the name of the keystore as it is referenced by the runtime. Other SSL configurations can reference this name further down in the `ssl.client.props` file to avoid duplication.
com.ibm.ssl.keyStore	${user.root}/etc/key.p12	This property specifies the location of the keystore in the required format of the com.ibm.ssl.keyStoreType property. Typically, this property references a keystore file name. However, for cryptographic token types, this property references a Dynamic Link Library (DLL) file.
com.ibm.ssl.keyStorePassword	WebAS	This property is the default password, which is the cell name for the profile when it is created. The password is typically encoded using an {xor} algorithm. You can use iKeyman to change the password in the keystore, then change this reference. If you do not know the password and if the certificate is created for you, change the password in this property, then delete the keystore from the location where it resides. Restart the client to recreate the keystore by using the new password, but only if the keystore name ends with DefaultKeyStore and if the fileBased property is `true`. Delete both the keystore and truststore at the same time so that a proper signer exchange can occur when both are recreated together.
com.ibm.ssl.keyStoreType	PKCS12	This property is the keystore type. Use the default, PKCS12, because of its interoperability with other applications. You can specify this property as any valid keystore type that is supported by the JVM on the provider list.

Property	Default	Description
com.ibm.ssl.keyStoreProvider	IBMJCE	The IBM Java Cryptography Extension property is the keystore provider for the keystore type. The provider is typically IBMJCE or IBMPKCS11Impl for cryptographic devices.
com.ibm.ssl.keyStoreFileBased	true	This property indicates to the runtime that the keystore is file-based, meaning it is located on the file system.
com.ibm.ssl.keyStoreReadOnly	false	This property indicates to the runtime whether the keystore can be modified during the runtime.

Truststore Configurations

SSL configurations reference truststore configurations, whose purpose is to contain the signer certificates for servers that are trusted by this client. You can specify these properties with other SSL configuration properties. However, it is recommended that you specify truststore configurations in this section of the `ssl.client.props` file after the com.ibm.ssl.trustStoreName property has identified the start of a new truststore configuration. After you fully define the truststore configuration, the com.ibm.ssl.trustStoreName property can reference the configuration at any other point in the file.

A truststore is a keystore that JSSE uses for trust evaluation. A truststore contains the signers that WebSphere Application Server requires before it can trust the remote connection during the handshake. If you configure the com.ibm.ssl.trustStoreName=ClientDefaultKeyStore property, you can reference the keystore as the truststore. Further configuration is not required for the truststore because all of the signers that are generated through signer exchanges are imported into the keystore where they are called by the runtime.

Property	Default	Description
com.ibm.ssl.trustStoreName	ClientDefaultTrustStore	This property specifies the name of the truststore as it is referenced by the runtime. Other SSL configurations can reference further down in the `ssl.client.props` file to avoid duplication.
com.ibm.ssl.trustStore	${user.root}/etc/trust.p12	This property specifies the location of the truststore in the format that is required by the truststore type that is referenced by the com.ibm.ssl.trustStoreType property. Typically, this property references a truststore file name. However, for cryptographic token types, this property references a DLL file.
com.ibm.ssl.trustStorePassword	WebAS	This property specifies the default password, which is the cell name for the profile when it is created. The password is typically encoded using an {xor} algorithm. You can use iKeyman to change the password in the keystore, then change the reference in this property. If you do not know the password and if the certificate was created for you, change the password in this property, then delete the truststore from the location where it resides. Restart the client to recreate the truststore by using the new password, but only if the keystore name ends with DefaultTrustStore and the fileBased property is `true`. It is recommended that you delete the keystore and the truststore at the same time so that a proper signer exchange can occur when both are recreated together.

Property	Default	Description
com.ibm.ssl.trustStoreType	PKCS12	This property is the truststore type. Use the default PKCS12 type because of its interoperability with other applications. You can specify this property as any valid truststore type that is supported by the JVM functionality on the provider list.
com.ibm.ssl.trustStoreProvider	IBMJCE	This property is the truststore provider for the truststore type. The provider is typically IBMJCE or IBMPKCS11Impl for cryptographic devices.
com.ibm.ssl.trustStoreFileBased	true	This property indicates to the runtime that the truststore is file-based, meaning it is located on the file system.
com.ibm.ssl.trustStoreReadOnly	false	This property indicates to the runtime whether the truststore can be modified during the runtime.

Creating a keystore configuration

A Secure Sockets Layer (SSL) configuration references keystore configurations during WebSphere Application Server runtime. Whether a keystore file was created by another keystore tool or saved from a previous configuration, the file must be part of a keystore configuration object. You can create a keystore configuration for the existing keystore object.

A keystore must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration* > Key stores and certificates > New**.
2. Type a name in the **Name** field. This name uniquely identifies the keystore in the configuration.
3. Type the location of the keystore file in the **Path** field. The location can be a file name or a file URL to an existing keystore file.
4. Type the keystore password in the **Password** field. This password is for the keystore file that you specified in the **Path** field.
5. Type the keystore password again in the **Confirm Password** field to confirm the password.
6. Select a keystore type from the list. The type that you select is for the keystore file that you specified in the **Path** field.
7. Select any of the following optional selections:
   - The **Read only selection** creates a keystore configuration object but does not create a keystore file. If this option is selected, the keystore file that you specified in the **Path** field must already exist.
   - The **Initialize at startup selection** initializes the keystore during runtime.
8. Click **Apply** and **Save**.

You have created a keystore configuration object for the keystore file that you specified. This keystore can now be used in an SSL configuration.

You can create additional keystore configurations, as needed.

Changing a keystore password

You can change the WebSphere Application Server password from the default password value, `WebAS`, in the administrative console or at a Java command prompt. Because the default password is well known, it is important that you change the password after you install WebSphere Application Server to protect the security of the keystore files and the Secure Sockets Layer (SSL) configuration.

When WebSphere Application Server starts for the first time as a standalone application server or as a Network Deployment Server, each server creates a keystore and truststore for the default SSL configuration. WebSphere Application Server creates these files and assigns the default password, `WebAS`.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Key Stores and Certificates**.
2. Select a keystore.
3. Type the new keystore password in the **Change Password** field.
4. Type the keystore password again in the **Confirm Password** field to confirm the password.
5. Select **Apply**.

WebSphere Application Server and its keystores are protected by a unique password.

Use the new password the next time you start the WebSphere Application Server.

Configuring a hardware cryptographic keystore

You can create a hardware cryptographic keystore that WebSphere Application Server can use to provide cryptographic token support in the server configuration.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration* > Key stores and certificates**.
2. Click **New**.
3. Type a name to identify the keystore. This name is used to enable hardware cryptography in the Web services security configuration.
4. Type the path for the hardware device-specific configuration file. The configuration file is a text file that contains entries in the following format: *`attribute = value`*. The valid values for attribute and value are described in detail in the Software Developer Kit, Java Technology Edition documentation. The two mandatory attributes are name and library, as shown in the following sample code:

   ```
   name = FooAccelerator
   library = /opt/foo/lib/libpkcs11.so
   slotListIndex = 0
   ```

5. Type a password if the token login is required. Operations that use keys on the token require a secure login. This field is optional if the keystore is used as a cryptographic accelerator. In this case, you need to select **Enable pure acceleration for hardware cryptographic operations**.
6. Select the **PKCS11** type.
7. Select **Read only**.
8. Click **OK** and **Save**.

WebSphere Application Server can now provide cryptographic token support in the server configuration.

You can refer to this keystore in any server Secure Sockets Layer (SSL) configuration to achieve the following results:

- Cryptographic acceleration because the cryptographic hardware device has no persistent key storage
- Secure cryptographic hardware because a cryptographic token generates and securely stores the private key that WebSphere Application Server uses for SSL key exchange.

You can also refer to this keystore in the Web services security default bindings configuration to achieve similar results.

Managing keystore configurations remotely

You can manage keystores remotely in a Network Deployment environment on separate machines. A node server can hold the configuration for a keystore, while the actual keystore resides on another system. After

you set up a remotely managed configuration, you can perform all of the certificate and keystore operations for the keystore on the remote machine from the server that contains the keystore remote configuration.

Key stores can be remotely managed only in network deployed environments.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates**.
2. Click **New**.
3. Type a name in the **Name** field. This name uniquely identifies the keystore in the configuration.
4. Type the location of the keystore file in the **Path** field. The location can be a file name or a file Uniform Resource Locator (URL) to an existing keystore file.
5. Type the keystore password in the **Password** field. This password is for the keystore file that you specified in the **Path** field.
6. Type the keystore password again in the **Confirm Password** field to confirm the password.
7. Select a keystore type from the list. The type you select is for the keystore file that you specified in the **Path** field.
8. Select the **Remotely managed** check box, and then fill in one or more hosts names of the systems where the keystore file is to be located. If you provide multiple host names, separate the host names with a pipe (**|**).
9. Select any of the following optional selections:
   - The **Read only selection** creates a keystore configuration object but does not create a keystore file. If this option is selected, the keystore file that you specified in the **Path** field must already exist.
   - The **Initialize at startup selection** initializes the keystore during run time.
10. Select **Apply** and **Save**.

A keystore configuration object is created on the server from where the command was run. The keystore file for the configuration will be created on each system that you specified in the host list.

Now, you can perform all certificate management operations on the keystore from the system where the keystore configuration resides. For example, you can perform certificate management operations, such as: creating a self-signed certificate, extracting a certificate, or extracting a signer certificate.

To manage a self-signed certificates by using the wsadmin tool, use the PersonalCertificateCommands group commands of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 1734.

Key stores and certificates collection

Use this page to manage key store types, including cryptography, Resource Access Control Facility (RACF) , Certificate Management Services (CMS), Java, and all trust store types.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound >** *SSL_configuration_name*. Under Related items, click **Key stores and certificates**.

Button	Resulting action
New	Adds a new key store object that can be referenced by Secure Sockets Layer (SSL) configurations or KeySets. The KeyStore management scope is based on the part of the topology tree from which it was created.

Button	Resulting action
Delete	Deletes an existing key store. The key store should not be referenced by any other parts of the configuration before you delete it.
Exchange signers	Refers to exchanging signers in a key store. You can select two key stores, along with personal certificates from a selected key store, then add them as a signer to another selected key store.

Name:

Specifies the unique name that is used to identify the key store. This name is typically scoped by the ManagementScope scopeName and based upon the location of the key store. The name must be unique within the existing key store collection.

This is a user-defined name.

Path:

Specifies the location of the key store file in the format needed by the key store type. This file can be a card-specific configuration file for cryptographic devices or a filename or file URL for file-based key stores. It can be a safkeyring URL for RACF keyrings.

Key store settings

Use this page to create all keystore types, including cryptographic, Resource Access Control Facility (RACF), Certificate Management Services (CMS), Java, and all truststore types.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations >** {**Inbound | Outbound** *SSL_configuration_name*}. Under Related Items, click **Key stores and certificates > New**.

Links to Personal certificates, Signer certificates, and Personal certificate requests enable you to manage certificates in a manner similar to iKeyman capabilities. A keystore can be file-based, such as CMS or Java keystore types, or it can be remotely managed.

Note: Any changes made to this panel are permanent.

Name:

Specifies the unique name to identify the keystore. The keystore is typically scoped by the ManagementScope scopeName based on the location of the keystore. The name must be unique within the existing keystore collection.

Data type: Text

Path:

Specifies the location of the keystore file in the format needed by the keystore type. This file can be a dynamic link library (DLL) for cryptographic devices or a filename or file URL for file-based keystores. It can be a safkeyring URL for RACF keyrings.

Data type: Text

Enable cryptographic operations on hardware device:

Specifies whether a hardware cryptographic device is used for cryptographic operations only. Operations that require a login are not supported when using this option.

Default: Disabled

Password [new keystore] / Change password [existing keystore]:

Specifies the password used to protect the keystore. For the default keystore (names ending in DefaultKeyStore or DefaultTrustStore), the password is the Cell name. This default password must be changed.

This field can be edited.

Data type: Text

Confirm password:

Specifies confirmation of the password to open the keystore file or device.

Data type: Text

Type:

Specifies the implementation for keystore management. This value defines the tool that operates on this keystore type.

The list of options is returned by java.security.Security.getAlgorithms(″KeyStore″). Some options might be filtered and some might be added based on the java.security configuration.

Data type: Text
Default: PKCS12

Read only:

Specifies whether the keystore can be written to or not. If the keystore cannot be written to, certain operations cannot be performed, such as creating or importing certificates.

Default: Disabled

Initialize at startup:

Specifies whether the keystore needs to be initialized before it can be used for cryptographic operations. If enabled, the keystore is initialized at server startup.

Default: Disabled

Key managers collection

Use this page to define the implementation settings for key managers. A key manager is invoked during a Secure Sockets Layer (SSL) handshake to determine which certificate alias is used. The default key manager (WSX509KeyManager) performs alias selection. If more advanced function is desired, define a custom key manager and select it on the **Manage endpoint security configurations** panel.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound >** *SSL_configuration_name*. Under Related items, click **Key managers**.

Button	Resulting action
New	Adds a new key manager that can be selected by an SSL configuration. A key manager is invoked during an SSL handshake to select a specific certificate alias to use from a key store.
Delete	Deletes an existing key manager. The key manager should not be referenced by any SSL configuration before you can delete it.

Name:

Specifies the name of the key manager, which you can select on the SSL configuration panel.

Class name:

Specifies the name of the key manager implementation class. This class implements javax.net.ssl.X509KeyManager interface and, optionally, the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface.

Algorithm:

Specifies the algorithm name of the key manager that is implemented by the selected provider.

Key managers settings

Use this page to define key managers implementation settings. A key manager gets invoked during an Secure Sockets Layer (SSL) handshake to determine the certificate alias to be used. The default key manager (WSX509KeyManager) performs alias selection. If more advanced function is desired, a custom key manager can be specified here and selected in the SSL configuration.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound >** *SSL_configuration_name*. Under Related items, click **Key managers > New**.

Name:

Specifies the name of the key manager, which you can select on the SSL configuration panel.

Data type: Text

Standard:

Specifies the key manager selection that is available from a Java provider that is installed in the java.security file. This provider might be shipped by Java Secure Sockets Extension (JSSE) or be a custom provider that implements an X509KeyManager interface.

Default: Enabled

Provider:

Specifies the provider name that has an implementation of an X509KeyManager interface. This provider is typically set to IBMJSSE2.

Data type: Text
Default: IBMJCE

Algorithm:

Specifies the algorithm name of the trust manager implemented by the selected provider.

Data type: Text
Default: IbmX509

Custom:

Specifies that the key manager selection is based on a custom implementation class that implements the javax.net.ssl.X509KeyManager interface and optionally the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface to obtain additional connection information not otherwise available.

Default: Disabled

Class name:

Specifies the name of the key manager implementation class.

Data type: Text

Creating a self-signed certificate

You can create a self-signed certificate. WebSphere Application Server uses the certificate at runtime during the handshake protocol. Self-signed certificates are located in the default keystore.

You must create a keystore before you can create a self-signed certificate.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > [*keystore*]**.
2. From Additional Properties, click **Personal certificates**.
3. Click **Create a self-signed certificate**.
4. Type a certificate alias name. The alias identifies the certificate request in the keystore.
5. Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).
6. Type an organization value. This value is the O value in the certificate DN.
7. You can configure one or more of the following optional values:
   a. **Optional:** Select a key size value. The default key size value is 1024 bits.
   b. **Optional:** Type an organizational unit value. This organizational unit value is the OU value in the certificate DN.
   c. **Optional:** Type a locality value. This locality value is the L value in the certificate DN.
   d. **Optional:** Type a state or providence value. This value is the ST value in the certificate DN.
   e. **Optional:** Type a zip code value. This zip code value is the POSTALCODE value in the certificate DN.
   f. **Optional:** Select a country value from the list. This country value is the C= value in the certificate request DN.

8. Click **Apply**.

You have created a self-signed certificate that resides in the keystore. The SSL configuration for the WebSphere Application Server runtime uses this certificate for SSL communication. Extract the signer of the self-signed certificate to add the signer to another keystore.

To create a self-signed certificate by using the wsadmin tool, use the **createSelfSignedCertificate** command of the AdminTask object. For more information, see "Commands for the PersonalCertificateCommands group of the AdminTask object" on page 1734.

Replacing an existing self-signed certificate

Occasionally, you need to replace an existing or expired self-signed certificate with a new certificate. Certificates are referenced in the runtime configuration by the Secure Sockets Layer (SSL) Configuration object and the Dynamic SSL Configuration Selection object. You can replace a certificate with a new certificate alias reference or with a new signer certificate.

The current certificate and the certificate replacement must exist in the same keystore before you can replace a certificate.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > [*keystore*]**.
2. Under Additional Properties, click **Personal certificates**.
3. Select a personal certificate. The alias list must include at least two certificates that reside in the keystore.
4. Click **Replace**.
5. Select a replacement certificate alias from the list.
6. You can delete one of the following types of certificates:
   - Select **Delete old certificate** to delete the existing certificate.
   - Select **Delete old signers** to delete the existing signer certificates.
7. Click **Apply**.

Your results depend on what you selected:

- If you selected **Delete old certificate**, the new certificate alias replaces all of the references to the certificate alias in the configuration.
- If you selected **Delete old signers**, the new signer certificate replaces all of the occurrences of the old signer certificates.
- If the new certificate alias replaces the existing alias, the WebSphere Application Server runtime checks to make sure that:
  - All of the SSL Configurations objects reference the certificate
  - The Dynamic SSL Configuration Selections objects and the SSL Configuration group objects reference the certificate.
- If you selected **Delete old signers**, the existing signer certificates are replaced.
- If you selected **Delete old certificate**, the existing certificate are deleted.

To replace a self-signed certificate by using the wsadmin tool, use the **replaceCertificate** command of the AdminTask object. For more information, see rxml_atpersonalcert.dita.

Creating a certificate authority request

To ensure Secure Sockets Layer (SSL) communication, servers require a personal certificate that is either self-signed or signed by a certificate authority (CA). You must first create a personal certificate request to obtain a certificate that is signed by a CA.

The keystore that contains a personal certificate request must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > *keystore***.
2. Click **Personal certificate requests > New**.
3. Type the full path of the certificate request file. The certificate request is created in this location.
4. Type an alias name in the **Key label** field. The alias identifies the certificate request in the keystore.
5. Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).
6. Type an organization value. This value is the O value in the certificate DN.
7. You can configure one or more of the following optional values:
   a. **Optional:** Select a key size value. The default key size value is 1024 bits.
   b. **Optional:** Type an organizational unit value. This organizational unit value is the OU value in the certificate DN.
   c. **Optional:** Type a locality value. This locality value is the L value in the certificate DN.
   d. **Optional:** Type a state or providence value. This value is the ST value in the certificate DN.
   e. **Optional:** Type a zip code value. The zip code value is the POSTALCODE value in the certificate DN.
   f. **Optional:** Select a country value from the list. This country value is the C= value in the certificate request DN.
8. Click **Apply**.

The certificate request is created in the specified file location in the keystore. The request functions as a temporary placeholder for the signed certificate until you manually receive the certificate in the keystore.

To create a certificate request using the wsadmin tool, use the **createCertificateRequest** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 1734.

Note: Key store tools (such as iKeyman and keyTool) cannot receive signed certificates that are generated by certificate requests from WebSphere Application Server. Similarly, WebSphere Application Server cannot accept certificates that are generated by certificate requests from other keystore utilities.

Now you can receive the CA-signed certificate into the keystore to complete the process of generating a signed certificate for your server.

Certificate request settings

Use this page to verify the properties of a personal certificate request.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store***. Under Additional Properties, click **Personal certificate requests > *certificate request*** .

Key label:

Specifies the certificate alias name for the signer in the key store, which is specified in the SSL configuration.

Key size:

Specifies the size of the keys that are generated.

Requested by:

Specifies the Subject distinguished name (DN) that represents the identity of the certificate request.

Fingerprint (SHA Digest):

Specifies the SHA hash of the personal certificate, which can be used to verify that the certificate has not been altered when it is used in a remote connection.

Signature algorithm:

Specifies the algorithm used to sign the certificate.

Personal certificates collection

Use this page to manage personal certificates.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificates**.

The **Personal certificates** page lists all Personal certificates in the selected key store. You can do most certificate management operations in this panel, including creating a new self-signed certificate, deleting a certificate, receiving one generated from a CA, replacing a certificate (simultaneous delete and create, replacing references across all key stores), extracting the signer, and importing or exporting a personal certificate.

Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA).

The Key store collection must contain at least two key store files. You must select one file in order to replace, extract, or export a key store,

Button	Resulting action
Create a self-signed certificate	Enables the application server to create a new self-signed certificate.
Delete	Specifies to delete a certificate from the key store. Be careful that the certificate alias is not referenced elsewhere in the Secure Sockets Layer configuration.
Receive a certificate from a certificate authority	Enables the application server to receive a certificate authority (CA)-generated certificate from a file to complete a certificate request.
Replace	Replaces a self-signed certificate with another self-signed certificate that contains the same information, but with a new expiration period. The signer from the old certificate that is contained in any managed key store in the cell is replaced by the signer from the new certificate.
Extract	Extracts a certificate from the key store that will be added to another key store as a trusted certificate (signer).
Import	Imports a certificate, including the private key, from a key store file.
Export	Exports a certificate, including the private key, to a specified key store file.

Alias:

Specifies the alias by which the personal certificate is referenced in the key store.

When you select an alias, the View Certificate panel opens.

Issued by:

Specifies the distinguished name of the entity by which the certificate was issued. This name is the same as the issued-to distinguished name when the personal certificate is self-signed.

Issued to:

Specifies the distinguished name of the entity to which the certificate was issued.

Serial number:

Specifies the certificate serial number that is generated by the issuer of the certificate.

Expiration:

Specifies the expiration date of the signer certificate for validation purposes.

Personal certificates settings

Use this page to create new personal certificates.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound >** *SSL_configuration_name*. Under Related items, click **Key stores and certificates >** *key store*. Under Additional Properties, click **Personal certificates > Create a self-signed certificate**.

This same help file is available when you create a new certificate or modify an existing certificate. The fields in this file are marked according to when they show on the administrative console.

Alias:

Specifies the alias for the personal certificate in the key store.

This field displays when you create a new certificate. This field is read-only when you view an existing certificate.

Data type: Text

Version:

Specifies the version of the personal certificate. Valid versions include X509 V3, X509 V2, or X509 V1. It is recommended to use X509 V3 certificates.

This field is read-only when you create or view a certificate.

Data type: Text
Default: X509 V3
Range:

Key size:

Specifies the key size of the private key that is used by the personal certificate.

This field displays when you create or view a certificate.

Data type: Integer
Default: 1024

Common name:

Specifies the common name portion of the distinguished name (DN). It is recommended that this name be the host name of the machine on which the certificate resides. In some cases, the common name is used to login during Secure Socket Layer (SSL) certificate authentication; therefore, in some cases, this name might be used as a user ID for a local operating system registry.

This field displays when you create a new certificate, but does not display when you view an existing certificate.

Data type: Text

Serial number:

Specifies the certificate serial number that is generated by the issuer of the certificate.

This field displays only when you view an existing certificate.

Validity period:

Specifies the length in days during which the certificate is valid. The default is 365 days.

This field displays when you create or view a certificate.

Data type: Text

Organization:

Specifies the organization portion of the distinguished name.

This field displays only when you create a new certificate.

Data type: Text

Organization unit:

Specifies the organization unit portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Text

Locality:

Specifies the locality portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Text

State/Province:

Specifies the state portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Text

Zip code:

Specifies the zip code portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Integer

Country or region:

Specifies the country portion of the distinguished name.

This field displays only when you create a new certificate.

Data type: Text
Default: (none)

Refer to http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html for a list of ISO 3166 country codes.

Validity period:

Specifies the length, in days, when the certificate is valid. The default is 365 days.

This read-only field displays only when you view an existing certificate.

Issued to:

Specifies the distinguished name of the entity to which the certificate was issued.

This read-only field displays only when you view an existing certificate.

Issued by:

Specifies the distinguished name of the entity that issued the certificate. When the personal certificate is self-signed, this name is identical to the **Issued to** distinguished name.

This read-only field displays only when you view an existing certificate.

Fingerprint (SHA Digest):

Specifies the Secure Hash Algorithm (SHA hash) of the certificate, which can be used to verify the certificate's hash at another location, such as the client side of a connection.

This read-only field displays only when you view an existing certificate.

Signature algorithm:

Specifies the algorithm used to sign the certificate.

This read-only field displays only when you view an existing certificate.

Personal certificate requests collection

Use this page to manage personal certificate requests. Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificate requests**.

A private key is generated during the certificate request generation, but only the certificate is sent to the CA. The CA generates a new certificate, signed by the CA. This can be added in the Personal Certificates panel.

Button	Resulting action
New	Creates a personal certificate request that can be given to a certificate authority to complete.
Delete	Deletes a personal certificate request.
Extract	Extracts a personal certificate request. Only one certificate request can be selected at a time.

Note: Any changes made to this panel are permanent.

Key label:

Specifies the alias that represents the personal certificate request in the key store.

Requested by:

Specifies the Subject distinguished name (DN) that represents the identity of the certificate request.

Personal certificate requests settings

Use this page to create a new certificate request that can be extracted and sent to a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificate requests > New**.

Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA). The private key is generated during the certificate request generation, but only the certificate is sent to the CA. The CA generates a new certificate, signed by the CA.

Note: Any changes made to this panel are permanent.

File for certificate request:

Specifies the fully qualified file name from which the certificate request is exported. This portion of the certificate request can be given to the certificate authority to generate the real certificate. After the real certificate is generated, you can perform an ″Receive a certificate from a certificate authority″ from the personal certificate collection view.

Data type: Text

Key label:

Specifies the alias that represents the personal certificate request in the key store.

Data type: Text

Key size:

Specifies the size of the keys that are generated.

Data type: Integer
Default: 1024

Common name:

Specifies the name of the entity that the certificate represents. This common name can represent a person, company, or machine. For Web sites, the common name is frequently the DNS host name where the server resides.

Data type: Text

Organization:

Specifies the organization portion of the distinguished name.

Data type: Text

Organizational unit:

Specifies the organization unit portion of the distinguished name. This field is optional.

Data type: Text

Locality:

Specifies the locality portion of the distinguished name. This field is optional.

Data type: Text

State/Province:

Specifies the state portion of the distinguished name. This field is optional.

Data type: Text

Zip code:

Specifies the zip code portion of the distinguished name. This field is optional.

Data type: Integer

Country or region:

Specifies the country portion of the distinguished name.

Data type: Text
Default: US

Refer to http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html for a list of ISO 3166 country codes.

Extract certificate request

Use this page to extract a certificate request and send it to a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificate requests > Extract**.

Key label:

Specifies the alias that represents the personal certificate request in the key store.

File for certificate request:

Specifies the filename where the extracted certificate request is placed.

Data type: Text

Receiving a certificate issued by a certificate authority

When a certificate authority (CA) receives a certificate request, it issues a new certificate that functions as a temporary placeholder for a CA-issued certificate. A keystore receives the certificate from the CA and generates a CA-signed personal certificate that WebSphere Application Server can use for Secure Sockets Layer (SSL) security.

The keystore must contain the certificate request that was created and sent to the CA. Also, the keystore must be able to access the certificate that is returned by the CA.

WebSphere Application Server can receive only those certificates that are generated by a WebSphere Application Server certificate request. It cannot receive certificates that are created with certificate requests from other keystore tools, such as **iKeyman** and **keyTool**.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates >** [*keystore*].
2. Under Additional Properties, click **Personal certificates**.

3. Select a personal certificate.
4. Click **Receive a certificate from a certificate authority**.
5. Type the full path and name of the certificate file.
6. Select a data type from the list.
7. Click **Apply** and **Save**.

The keystore contains a new personal certificate that is issued by a CA. The original certificate request is changed to a personal certificate.

The SSL configuration is ready to use the new CA-signed personal certificate.

To receive a certificate by using the wsadmin tool, use the **receiveCertificate** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 1734.

Export certificate to a key file:

Use this page to specify a personal certificate to export to a key file.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificates > Export**.

Certificate alias to export:

Displays the name of the certificate that you selected to export on the previous panel.

Data type: Text

Alias:

Specifies the alias that the personal certificate is referenced by in the key store.

Data type: Text

Key file name:

Specifies the key store file name into which the exported certificate is added. If the key store file name already exists, the exported certificate will be added. If the key store file name does not already exist, one will be created, and the exported certificate will be added.

Data type: Text

Type:

Specifies the type of key store file. The valid types are listed in the menu.

Data type: Text
Default: JKS

Key file password:

Specifies the password that is used to access the key store file.

Data type: Text

Confirm password:

Confirms the password entered in the previous field.

The key file password and the new alias must be specified in case a certificate already exists with the same name.

Data type: Text

Import certificate from a key file:

Use this page to specify a personal certificate to import from a key file.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound >** *SSL_configuration_name*. Under Related items, click **Key stores and certificates >** ***key store*** . Under Additional Properties, click **Personal certificates > Import certificates from a key file**.

Key file name:

Specifies the fully qualified path to keystore file that contains the certificate to import.

Data type: Text

Type:

Specifies the type of keystore file. The valid types are listed in the menu.

Data type: Text

Key file password:

Specifies the password that is used to access the keystore file.

Data type: Text

Certificate alias to import:

Specifies the certificate alias identified as the **Key file name** that you want to import into the current key store.

Data type: Text
Default: (none)

Imported certificate alias:

Specifies the new alias that you want the certificate to be named in the current key store.

Data type: Text

Receive certificate from CA:

Use this page to import your personal certificate from the certificate authority (CA). The imported certificate replaces the temporary certificate associated with the public/private keys in the certificate request that is stored in the key store.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificates > Receive certificate from CA**.

Certificate file name:

Specifies the filename that contains the certificate generated by the certificate authority (CA).

Data type: Text

Data type:

Specifies the format of the file that is either Base64 encoded ASCII data or Binary DER data.

Data type: Text
Default: Base64-encoded ASCII data

Replace a certificate

Use this page to specify two certificates: the first selected certificate is replaced by the second selected certificate. The replace function replaces all the old signer certificates in key stores that are managed throughout the cell with the new signer from the new certificate. The same level of trust that was established with the old certificate is maintained.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties click **Personal certificates > Replace certificate**.

Old certificate:

Specifies the certificate that you want to replace.

Data type: Text

Replace with:

Specifies the certificate that you want to replace the old certificate.

Data type: Text
Default: (none)

Delete old certificate after replacement:

Specifies that you want to delete the old certificate and all associated signer certificates after the new certificate replaces it. If you do not replace the old personal certificate, it might be assigned a new alias name.

Default: Disabled

Delete old signers:

Specifies that you want to delete the old signer certificates that are associated with the old certificate after the new signer certificates replace them. If you do not delete the old signer certificates, they might be assigned new alias names.

Default: Disabled

Extracting a signer certificate from a personal certificate

Personal certificates contain a private key and a public key. You can extract the public key, called the *signer certificate*, to a file, then import the certificate into another keystore. The client requires the signer portion of a personal certificate for Security Socket Layer (SSL) communication.

The keystore that contains a personal certificate must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates > *keystore*** .
2. Under Additional Properties, click **Personal certificates**.
3. Select a personal certificate.
4. Click **Extract**.
5. Type the full path for the certificate file name. The signer certificate is written to this certificate file.
6. Select a data type from the list.
7. Click **Apply**.

The signer portion of the personal certificate is stored in the file that is provided.

This signer can now be imported into other keystores.

To extract a signer certificate from a personal certificate using the wsadmin tool, use the **extractCertificate** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 1734.

Extract certificate

Use this page to extract the signer from the personal certificate. The certificate can be imported into a trust store for trust verification.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store*** . Under Additional Properties, click **Personal certificates > Extract**.

Certificate alias to extract:

Specifies the filename that contains the extracted certificate.

Data type: Text

Certificate file name:

Specifies the fully qualified path where the certificate file will reside.

Data type: Text

Data type:

Specifies the format of the file, which is either Base64-encoded ASCII data or Binary DER data.

Data type: Text
Default: Base64-encoded ASCII data

Extract signer certificate

Use this page to extract a signer certificate that can be added elsewhere.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > Key stores and certificates > *key store* > Signer Certificates > Extract signer certificate**.

File name:

Specifies the fully qualified file name where the extracted signer certificate is placed.

Data type: Text

Data type:

Specifies the format of the file, which is either Base64 encoded ASCII data or Binary DER data.

Data type: Text

Retrieving signers using the retrieveSigners utility at the client

The client requires the signer certificates from the server to be able to communicate with WebSphere Application Server. Use the **retrieveSigners** command to get the signer certificate from a server.

The retrieveSigners utility is located in one of the following directories, depending on your operating system:

- On Windows operating system: *profile_root*\bin . For example: C:\WebSphere\AppServer\profiles\AppSrv01\bin
- On UNIX operating systems: ../*profile_root*/bin

In this release, a Java client that does not have access to a stdin console prompt should use the retrieveSigners utility to download the signers from the remote server key store when signers are needed for a Secure Sockets Layer (SSL) handshake. For example, you might interpret the client as failing to respond if an applet client or Java Web Start Client application cannot access the stdin signer exchange prompt. Thus, you must add the WebSphere Java method call **com.ibm.wsspi.ssl.RetrieveSignersHelper.callRetrieveSigners** to your client application to retrieve the signers and to avoid running the retrieveSigners utility manually.

Use the retrieveSigners utility for situations where you cannot verify whether or not the com.ibm.ssl.enableSignerExchangePrompt= property is enabled or disabled when the application makes a request. Set the com.ibm.ssl.enableSignerExchangePrompt= property to `false` in the ssl.client.props file if you cannot see the console.

Alternatively, you can manually create the server key in the client truststore.

Complete the following steps, as required:

1. Use the **retrieveSigners** command to get the signer certificate from a server. You can find details about the **retrieveSigners.** parameters in “Secure installation for client signer retrieval” on page 1418.
2. If the client and server are on the same machine, you will need only the *remoteKeyStoreName* and *localKeyStoreName* parameters. The most typical key store to reference on a remote system is `CellDefaultTrustStore` on a network deployed environment and `NodeDefaultTrustStore` on an application server.
3. When retrieving signers from a remote server, add these required connection-related parameters: **–host** *host*, **–port** *port*, **–conntype** {*RMI* | *SOAP*}.
4. If the remote server has security enabled, also supply the **–user** *user* **–password** *password* parameters.
5. Use the **–autoAcceptBootstrapSigner** parameter if you want to enable automation of the signer retrieval. This parameter automatically adds to the server all the signers that are needed to make the connection.

After running, the command displays the SHI-1 digest of the signers added. The output looks similar to the following output:

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin\retrieveSigners.bat
CellDefaultTrustStore ClientDefaultTrustStore

CWPKI0308I: Adding signer alias "default_signer" to local keystore
           "ClientDefaultTrustStore" with the following SHA digest:
```

See the following examples of how to call the `retrieveSigners.bat` file on the Windows operating system:

To retrieve signers on the same system, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
```

To retrieve signers on a remote system with a SOAP connection, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaulTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 8879 -conntype SOAP -autoAcceptBootstrapSigner
```

To retrieve signers on a remote system with an RMI connection, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 2809 -conntype RMI -autoAcceptBootstrapSigner
```

To retrieve signers on a remote system that has security enabled, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 8879 -conntype SOAP -user testuser -password testuserpwd
-autoAcceptBootstrapSigner
```

Changing the signer auto-exchange prompt at the client

For clients to communicate with WebSphere Application Server, clients must obtain a signer certificate from the server. Clients can use the **retrieveSigners** command to connect to a server to obtain the appropriate signer. A prompt displays that asks whether or not you want to add a signer to the truststore. If the Secure Sockets Layer (SSL) configuration uses an automated script that might hang, use the prompt to obtain the certificate.

The com.ibm.ssl.enableSignerExchangePrompt property in the *profile_home*/properties/ssl.client.props file controls the signer certificate prompt. By default, this property is set to true, meaning the prompt is enabled.

Complete the following steps to disable or enable the signer-exchange prompt at the client:

1. Open the *profile_home*/properties/ssl.client.props file using an editor.
2. Locate the section containing the SSL configuration information for the client that you are working with.
3. Change the value of the com.ibm.ssl.enableSignerExchangePrompt property to false if you do not want the signer-exchange prompt, or set it to true if you want to be prompted.
4. Save and close the file.

When the com.ibm.ssl.enableSignerExchangePrompt property is set to false, no prompt displays, so you can exchange signers while some administrative clients are running. When the com.ibm.ssl.enableSignerExchangePrompt property is set to true, a signer-exchange prompt displays, and you are asked to accept or reject the certificate. The prompt looks like the following example:

```
C:\WebSphere\AppServer\profiles\dmgr\bin>serverStatus -all
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Dmgr\logs\serverStatus.log
ADMU0128I: Starting tool with the dmgr profile
ADMU0503I: Retrieving server status for all servers
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr

*** SSL SIGNER EXCHANGE PROMPT ***
SSL signer from target host 192.174.1.5 is not found in truststore
C:/WebSphere/AppServer/profiles/Dmgr/etc/trust.p12.
```

Verify that the digest value matches what is displayed at the server in the following signer information:

```
Subject DN:    CN=hostname.austin.ibm.com, O=IBM, C=US
Issuer DN:     CN=hostname.austin.ibm.com, O=IBM, C=US
Serial number: 1128544457
Expires:       Thu Oct 20 15:34:17 CDT 2006
SHA-1 Digest:  91:A1:A9:2D:F2:7D:70:0F:04:06:73:A3:B4:A4:9C:56:9D:A8:A3:BA
MD5 Digest:    88:72:C5:88:00:1C:A7:FA:D6:EB:04:88:AC:A1:C9:13

Add signer to the truststore now? (y/n) y
A retry of the request might need to occur.
ADMU0508I: The Application Server "server1" is STARTED.
```

Clients can instigate communications for various processes using signer certificates obtained from WebSphere Application Server.

Importing a signer certificate from a truststore to a z/OS keyring

You can import a signer certificate, which is also called a certificate authority (CA) certificate, from a truststore on a non-z/OS platform server to a z/OS keyring.

To import a certificate to a z/OS keyring, complete the following steps:

1. On the non-z/OS platform server, change to the install_root/bin directory and start the iKeyman utility, which is called ikeyman.bat (Windows) or ikeyman.sh (UNIX). The *install_root* variable refers to the installation path for WebSphere Application Server.
2. Within the iKeyman utility, open the server truststore. The default server truststore is called the DummyServerTrustFile.jks file. The file is located in the $[USER_INSTALL_ROOT}/etc/ directory. The default password is WebAS.
3. Extract the signer certificate from the truststore using the **ikeyman** utility. Complete the following steps to extract the signer certificate:
   a. Select **Signer certificates** from the menu.
   b. Select **new websphere dummy server alias**.

c. Select **Extract**.
d. Select the correct data type. The signer_certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
e. Specify the fully qualified path and the file name of the certificate.

4. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.
5. From an FTP prompt on the non-z/OS platform server, type `put 'signer_certificate' mvs.dataset`. The *signer_certificate* variable refers to the name of the signer certificate on the non-z/OS platform server. The *mvs.dataset* variable is the data set name to which the certificate was exported.

   The `RACDCERT CERTAUTH ADD` command in the next step works with a Multiple Virtual Storage (MVS) data set only. You can either turn the certificate file into a binary MVS data set or use the put command with an Hierarchical File System (HFS) file, and then use the following command to copy the file into a MVS data set:

   ```
   OGET file_name mvs.dataset
   ```
6. On the z/OS platform server, go to option 6 in the Interactive System Productivity Facility (ISPF) dialog panels and issue the following commands as a super user to add the signer certificate to the z/OS keyring:
   a. Type `RACDCERT CERTAUTH ADD ('signer_certificate') WITHLABEL('Dummy Server CA') TRUST` The Dummy Server CA variable refers to the label name for the certificate authority (CA) certificate that you are importing from a non-z/OS platform server. The *keyring_name* variable refers to the name of the z/OS keyring that is used by the servers in the cell.
   b. Type `RACDCERT ID(ASCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)`
   c. Type `RACDCERT ID(DMCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)`
   d. Type `RACDCERT ID(DMSR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)` In the previous commands, `ASCR1`, `DMCR1`, and `DMSR1` are the RACF IDs under which the started tasks for the cell run in WebSphere Application Server for z/OS. The `ASCR1` value is the RACF ID for the application server control region. The `DMCR1` value is the RACF ID for the deployment manager control region. The `DMSR1` value is the RACF ID for the deployment manager server region.

After completing these steps, the z/OS keyring contains the signer certificates that originated on the non-z/OS platform server.

To verify that the certificates were added, use option 6 on the ISPF dialog panel and type the following command:

```
RACDCERT ID(CBSYMSR1) LISTRING(keyring_name)
```

The `CBSYMSR1` value is the RACF ID for the application server region.

Note: Although iKeyman is supported for WebSphere Application Server Version 6.1, customers are encouraged to use the administrative console to export signer certificates.

Exporting a signer certificate from WebSphere Application Server for z/OS to a truststore

You can export a signer certificate, which is also called a certificate authority (CA) certificate, from WebSphere Application Server for z/OS to a truststore.

WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express can use the certificate in the truststore.

To export the certificate to a truststore, complete the following steps:

1. Export the z/OS® signer certificate to a data set by issuing the following Resource Access Control Facility (RACF) command as a super user using Time Sharing Option (TSO) option 6:

   ```
   RACDCERT CERTAUTH EXPORT(LABEL('signer_certificate')) DSN('mvs.dataset')FORMAT(CERTDER)
   ```

The *signer_certificate* variable is the RACF label name of the certificate that is used by the cell. The *signer_certificate* can have either a Base64-encoded ASCII data type or a Binary DER data type. The *mvs.dataset* variable is the data set name to which the certificate is exported. You do not need to pre-allocate this data set because it is created by RACF.

2. From a command line on the non-z/OS platform server, type `cd` and change to the following directory: *install_root*`/profiles/default/etc`
3. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.
4. From an FTP prompt on the non-z/OS platform server, type the following command: `get '`*mvs.dataset*`' `*signer_certificate*
5. On the non-z/OS platform server, change to the *install_root*`/bin` directory and start the **iKeyman** utility, which is called `ikeyman.bat` for Windows or `ikeyman.sh` for UNIX.
6. Within the iKeyman utility, open the server truststore. The default server truststore is called the `DummyServerTrustFile.jks` file. The file is located in the `${USER_INSTALL_ROOT}/etc/` directory. The default password is `WebAS`. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
7. Add your exported signer certificate to the server truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
   a. Select **Signer certificates** from the menu.
   b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
   c. Specify the fully qualified path and file name of the signer certificate.
8. Within the iKeyman utility, open the client truststore. The default client truststore is called the `DummyClientTrustFile.jks` file. The file is located in the ${USER_INSTALL_ROOT}/etc/ directory. The default password is `WebAS`. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
9. Add your exported signer certificate to the client truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
   a. Select **Signer certificates** from the menu.
   b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
   c. Specify the fully qualified path and file name of the signer certificate.
10. Restart the server process to use the new signer certificates.

After completing these steps, you can use the exported signer certificates with the WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express products.

Importing a signer certificate from a truststore to a z/OS keyring

You can import a signer certificate, which is also called a certificate authority (CA) certificate, from a truststore on a non-z/OS platform server to a z/OS keyring.

To import a certificate to a z/OS keyring, complete the following steps:

1. On the non-z/OS platform server, change to the `install_root/bin` directory and start the iKeyman utility, which is called `ikeyman.bat` (Windows) or `ikeyman.sh` (UNIX). The *install_root* variable refers to the installation path for WebSphere Application Server.
2. Within the iKeyman utility, open the server truststore. The default server truststore is called the `DummyServerTrustFile.jks` file. The file is located in the `$[USER_INSTALL_ROOT}/etc/` directory. The default password is `WebAS`.
3. Extract the signer certificate from the truststore using the **ikeyman** utility. Complete the following steps to extract the signer certificate:
   a. Select **Signer certificates** from the menu.

b. Select **new websphere dummy server alias**.
c. Select **Extract**.
d. Select the correct data type. The signer_certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
e. Specify the fully qualified path and the file name of the certificate.

4. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.
5. From an FTP prompt on the non-z/OS platform server, type `put 'signer_certificate' mvs.dataset`. The *signer_certificate* variable refers to the name of the signer certificate on the non-z/OS platform server. The *mvs.dataset* variable is the data set name to which the certificate was exported.

   The `RACDCERT CERTAUTH ADD` command in the next step works with a Multiple Virtual Storage (MVS) data set only. You can either turn the certificate file into a binary MVS data set or use the put command with an Hierarchical File System (HFS) file, and then use the following command to copy the file into a MVS data set:

   ```
   OGET file_name mvs.dataset
   ```
6. On the z/OS platform server, go to option 6 in the Interactive System Productivity Facility (ISPF) dialog panels and issue the following commands as a super user to add the signer certificate to the z/OS keyring:
   a. Type `RACDCERT CERTAUTH ADD ('signer_certificate') WITHLABEL('Dummy Server CA') TRUST` The Dummy Server CA variable refers to the label name for the certificate authority (CA) certificate that you are importing from a non-z/OS platform server. The *keyring_name* variable refers to the name of the z/OS keyring that is used by the servers in the cell.
   b. Type `RACDCERT ID(ASCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)`
   c. Type `RACDCERT ID(DMCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)`
   d. Type `RACDCERT ID(DMSR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)` In the previous commands, `ASCR1`, `DMCR1`, and `DMSR1` are the RACF IDs under which the started tasks for the cell run in WebSphere Application Server for z/OS. The `ASCR1` value is the RACF ID for the application server control region. The `DMCR1` value is the RACF ID for the deployment manager control region. The `DMSR1` value is the RACF ID for the deployment manager server region.

After completing these steps, the z/OS keyring contains the signer certificates that originated on the non-z/OS platform server.

To verify that the certificates were added, use option 6 on the ISPF dialog panel and type the following command:

```
RACDCERT ID(CBSYMSR1) LISTRING(keyring_name)
```

The `CBSYMSR1` value is the RACF ID for the application server region.

Note: Although iKeyman is supported for WebSphere Application Server Version 6.1, customers are encouraged to use the administrative console to export signer certificates.

Exporting a signer certificate from WebSphere Application Server for z/OS to a truststore

You can export a signer certificate, which is also called a certificate authority (CA) certificate, from WebSphere Application Server for z/OS to a truststore.

WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express can use the certificate in the truststore.

To export the certificate to a truststore, complete the following steps:

1. Export the z/OS® signer certificate to a data set by issuing the following Resource Access Control Facility (RACF) command as a super user using Time Sharing Option (TSO) option 6:

   ```
   RACDCERT CERTAUTH EXPORT(LABEL('signer_certificate')) DSN('mvs.dataset')FORMAT(CERTDER)
   ```

The *signer_certificate* variable is the RACF label name of the certificate that is used by the cell. The *signer_certificate* can have either a Base64-encoded ASCII data type or a Binary DER data type. The *mvs.dataset* variable is the data set name to which the certificate is exported. You do not need to pre-allocate this data set because it is created by RACF.

2. From a command line on the non-z/OS platform server, type `cd` and change to the following directory:
   `install_root/profiles/default/etc`
3. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.
4. From an FTP prompt on the non-z/OS platform server, type the following command:
   `get 'mvs.dataset' signer_certificate`
5. On the non-z/OS platform server, change to the *install_root*`/bin` directory and start the **iKeyman** utility, which is called `ikeyman.bat` for Windows or `ikeyman.sh` for UNIX.
6. Within the iKeyman utility, open the server truststore. The default server truststore is called the `DummyServerTrustFile.jks` file. The file is located in the `${USER_INSTALL_ROOT}/etc/` directory. The default password is `WebAS`. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
7. Add your exported signer certificate to the server truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
   a. Select **Signer certificates** from the menu.
   b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
   c. Specify the fully qualified path and file name of the signer certificate.
8. Within the iKeyman utility, open the client truststore. The default client truststore is called the `DummyClientTrustFile.jks` file. The file is located in the ${USER_INSTALL_ROOT}/etc/ directory. The default password is `WebAS`. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
9. Add your exported signer certificate to the client truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
   a. Select **Signer certificates** from the menu.
   b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
   c. Specify the fully qualified path and file name of the signer certificate.
10. Restart the server process to use the new signer certificates.

After completing these steps, you can use the exported signer certificates with the WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express products.

Retrieving signers from a remote SSL port

To perform Secure Sockets Layer (SSL) communication with a server, WebSphere Application Server must retrieve a signer certificate from a secure remote SSL port during the handshake. After the signer certificate is retrieved, you can add the signer certificate to a keystore.

The keystore that is to contain the signer certificate must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > Key stores and certificates > *keystore* > Signer certificates > Retrieve from port**.
2. Click **Retrieve from port**.
3. Type the host name of the machine on which the signer resides.

4. Type the port location on the host machine on which the signer resides. The port location is not limited to ports on WebSphere Application Server. The ports can include Lightweight Directory Access Protocol (LDAP) ports or ports on any server on which an SSL port is already configured, such as `SIB_ENDPOINT_SECURE_ADDRESS`.
5. Select an SSL configuration for the outbound connection from the list.
6. Type an alias name for the certificate.
7. Click **Retrieve signer information**. A message window displays information about the retrieved signer certificate, such as: the serial number, issued-to and issued-by identities, SHA hash, and expiration date.
8. Click **Apply**. This action indicates that you accept the credentials of the signer.

The signer certificate that is retrieved from the remote port is stored in the keystore.

An SSL configuration or client process that requires an SSL connection to the server can use the retrieved and approved signer certificate.

To retrieve a signer certificate from a port using the wsadmin tool, use the **retrieveSignerFromPort** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 1734.

Retrieve from port

Use this page to retrieve a signer certificate from a remote SSL port. The system connects to the specified remote SSL host and port and receives the signer during the handshake using a trust manager. The signer SHA hash displays for validation and, if approved by an administrator, is added to the currently selected trust store.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store* > Signer certificates > Retrieve from port**.

Host:

Specifies the host name to which you connect when attempting to retrieve the signer certificate from the Secure Sockets Layer (SSL) port.

Data type: Text

Port:

Specifies the SSL port to which you connect when attempting to retrieve the signer certificate.

Data type: Text

SSL configuration for outbound connection:

Specifies the SSL configuration that is used to connect to the previously specified SSL port. This configuration is also the SSL configuration that contains the signer after retrieval. This SSL configuration does not need to have the trusted certificate for the SSL port as it is retrieved during validation and presented here.

Data type: Text
Default: DefaultSSLConfig

Alias:

Specifies the certificate alias name that you want to reference the signer in the key store, which is specified in the SSL configuration.

Data type: Text

Retrieved signer information:

Specifies the signer certificate information if it is retrieved from the remote host and port.

Serial number:

Specifies the certificate serial number that is generated by the issuer of the certificate.

Issued to:

Specifies the distinguished name of the entity to which the certificate was issued.

Issued by:

Specifies the distinguished name of the entity that issued the certificate. This name is the same as the issued-to distinguished name when the signer certificate is self-signed.

Fingerprint (SHA Digest):

Specifies the Secure Hash Algorithm (SHA hash) of the certificate, which can be used to verify the certificate's hash at another location, such as the client side of a connection.

Expiration:

Specifies the expiration date of the retrieved signer certificate for validation purposes.

Adding a signer certificate to a keystore

Signer certificates establish the trust relationship in SSL communication. You can extract the signer part of a personal certificate from a keystore, and then you can add the signer certificate to other keystores.

The keystore that you want to add the signer certificate to must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name* > Key stores and certificates**.
2. Select a keystore from the list of keystores.
3. Click **Add signers**.
4. Enter an alias for the signer certificate in the **Alias** field
5. Enter the full path to the signer certificate file in the **File name** field.
6. Select a data type from the list in the **Data type** field.
7. Click **Apply**.

When these steps are completed, the signer from the certificate file is stored in the keystore. You can see the signer in the keystore files list of signer certificates. Use the keystore to establish trust relationships for the SSL configurations.

To add a signer certificate to a keystore by using the wsadmin tool, use the **addSignerCertificate** command of the AdminTask object. For more information, see rxml_atpersonalcert.dita.

Add signer certificate

Use this page to add a signer certificate to the key store.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items click **Key stores and certificates > *key store* > Signer certificates > Add**.

Alias:

Specifies the alias that the signer certificate is referenced by in the key store.

Data type: Text

File name:

Specifies the filename where the encoded signer certificate is located.

Data type: Text

Data type:

Specifies the format of the file, which is either Base64 encoded ASCII data or Binary DER data.

Data type: Text

Signer certificates collection

Use this page to manage signer certificates in key stores. Signer certificates are used by Java Secure Socket Extensions (JSSE) to validate certificates sent by the remote side of the connection during a Secure Sockets Layer (SSL) handshake. If a signer does not exist in the trust store that can validate the certificate sent, the handshake fails and generates an ″unknown certificate″ error.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items click **Key stores and certificates > *key store* > Signer certificates**.

Button	Resulting action
Add	Adds a new trusted (signer) certificate.
Delete	Deletes an existing signer certificate.
Extract	Extracts a signer certificate from a personal certificate to a file.
Retrieve from port	Makes a test connection to an SSL port and retrieves the signer from the server during the handshake. The information from the certificate will be displayed so you can decide whether to trust it based upon the MD5 and/or SHA hash.

Alias:

Specifies the alias for this signer certificate in the key store.

Issued to:

Specifies the distinguished name of the entity that requested the certificate.

Fingerprint (SHA digest):

Specifies the Secure Hash Algorithm (SHA hash) of the certificate. This can be used to verify the hash for the certificate at another location, such as the client side of a connection.

Expiration:

Specifies the expiration date of the signer certificate for validation purposes.

Signer certificate settings

Use this page to verify the general properties of the selected signer certificate.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key stores and certificates > *key store* > Signer certificates > *signer certificate*** .

Alias:

Specifies the alias for this signer certificate in the key store.

Version:

Specifies the version of the personal certificate. Valid versions include X509 V3, X509 V2, or X509 V1.

Key size:

Specifies the key size of the public key used by the signer certificate.

Serial number:

Specifies the certificate serial number that is generated by the issuer of the certificate.

Validity period:

Specifies the begin and end dates of the certificate.

Issued to:

Specifies the distinguished name of the entity that requested the certificate.

Issued by:

Specifies the distinguished name of the entity that issued the certificate. This name is the same as the issued-to distinguished name when the signer certificate is self-signed.

Fingerprint (SHA Digest):

Specifies the Secure Hash Algorithm (SHA) hash of the certificate, which can be used to verify the hash for the certificate at another location such as the client side of a connection.

Signature algorithm:

Specifies the algorithm that is used to sign the certificate.

Exchanging signer certificates

To establish trust relationships, you can exchange signer certificates between keystores. When you exchange signer certificates, you are extracting a personal certificate from one keystore and adding it to another keystore as a *signer certificate*.

To exchange signer certificates, there must be two keystores.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key stores and certificates**.
2. Select two keystores from the list of keystores.
3. Click **Exchange signers**.
4. Select any of the certificates in the first personal certificates list, and click **Add**. After adding, the signer part of the selected personal certificate appears in the other (second) keystore signers list.
5. Select any of the certificates in the second personal certificates list, and click **Add**. After adding, the signer part of the selected personal certificate appears in the other (first) keystore signers list.
6. **Optional:** If you need to remove any of the certificates from either of the signers lists, highlight one or more of the certificates, and click **Remove**.
7. Click **Apply** and **Save**.

The signer certificate appears in the list for each keystore.

The extracted signer certificate is available to both keystores during the connection handshake.

Key stores and certificates exchange signers

Use this page to extract a personal certificate from one keystore and add it to another keystore as a signer certificate.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration***. Under Related items, click **Key stores and certificates > Exchange signers**.

Note: Any changes made to this panel are permanent.

[key store] personal certificates:

Specifies the personal certificates that are currently stored in the specified key store.

Press and hold the **Ctrl** key to select more than one item from the list.

Data type: Text

[key store] signers:

Specifies the trusted signer certificates that are currently stored in the specified key store and selected for the exchange.

Press and hold the **Ctrl** key to select more than one item from the list.

Data type: Text

Add:

Specifies to extract the signer from the selected personal certificate in the key store list on the left and add it to the signers list of the key store on the right.

After the certificate is added, it no longer displays in the left-hand list. The personal certificate is still in the key store, but it is no longer selectable

Remove:

Specifies to remove a selected signer from the signers list of the key store on the right. The removed certificate displays in the key store list on the left.

Configuring certificate expiration monitoring

When certificates expire, they can no longer be used by the system. WebSphere Application Server provides a utility to monitor certificates that are close to expiration or have already expired. You can schedule certificate monitoring, or you can request certificate monitoring on demand. You can also configure options for deleting expired certificates and for recreating certificates.

WebSphere Application Server notifies you when a certificate is about to expire. Complete the information required for notification messaging in “Notifications” on page 1500.

Complete the following configuration steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage certificate expiration**.
2. Type a number for the number of days threshold in the **Expiration notification threshold** field. WebSphere Application Server issues an expiration warning *n* number of days before expiration.
3. Select or check one or more of the following options:
   - **Expiration check notification**. Select the method from the list that you want to use to receive your notification.
   - **Automatically replace expiring self-signed certificates**. If you do not want to recreate the self-signed certificate, clear the check box.
   - **Delete expiring certificates and signers after replacement**. If you do not want to delete the expired certificates and signers, clear the check box.
   - **Enable checking**. If you do not want to have certificate monitoring enabled, clear the check box.
4. Enter the time of day when you want certificate monitoring to take place to schedule the running of the certificate expiration monitor.
5. Select one of the following options:
   - **Check by calendar**. For **Weekday**, enter the day of week that you want to run the certificate expiration monitor. For **Repeat Interval**, specify the frequency to run the certificate monitor.
   - **Check by number of days**. Enter a number for how frequently the monitor runs, in number of days.
6. Click **Apply**.

After completing the settings, a certificate expiration monitor object and a schedule are set up in the configuration. The certificate expiration monitor runs according to the configurations options that you configured.

You can generate reports that state which certificates have expired. The reports identify the notifications of certificate replacements and deletions. The report is sent according to the notification option that you specified.

Manage certificate expiration settings

Use this page to configure the certificate expiration monitor.

To view this administrative console page, click **Security > SSL certificate and key management > Manage certificate expiration**.

Attention: To see the changes to the Expiration checking fields, you must click **Apply**.

Expiration notification threshold:

Specifies a threshold number of days during which the application warns specified individuals that a certificate is about to expire. For example, when the expiration monitor is run and the threshold is 30 days, if the current date is 30 days or less from the certificate expiration date, the certificate is flagged for notification. The application server can be configured to provide certification expiration notification through either e-mail or the message log file.

Data type: Integer
Default: 30 days

Expiration check notification:

Specifies the notification type (such as e-mail or System Out) when an expiration monitor runs.

Default:

Automatically replace expiring self-signed certificates:

Specifies a new self-signed certificate be generated using the same certificate information if the expiration notification threshold is reached. The old certificate is replaced and uses the same alias. All old signers are managed by the key store configuration are also replaced. The system only replaces self-signed certificates.

Default: Enabled

Delete expiring certificates and signers after replacement:

Specifies whether to completely remove old, self-signed certificates from the key store during a replace operation or leave them there under a renamed alias. If an old certificate is not deleted, the system renames the alias so that the new certificate can use the old alias, which might be referenced elsewhere in the configuration.

Default: Enabled

Enable checking:

Specifies the certificate monitor is active and will run as scheduled.

Scheduled time of day to check for expired certificates:

Specifies the scheduled time that the system checks for expired certificates.

You can type the scheduled time in hours and minutes, specify either A.M. or P.M., or 24-hour.

Data type Integer
Default: 0, 0
Range: 1–12, 0–59

Check by calendar:

Indicates that you want to schedule a specific day of the week on which the expiration monitor runs. For example, it might run on Sunday.

Default: Disabled

Weekday:

Specifies the day of the week on which the expiration monitor runs if **Check on a specific day** is selected.

Default: Sunday
Range: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Repeat interval:

Specifies the period of time between each schedule time to check for expired certificates or the interval between schedule checks.

Default: Daily
Range: Daily, Weekly

Check by number of days:

Specifies that you want to schedule a specific number of days between each run of the expiration monitor. The day of the week on which this occurs is not counted. For example, if you set the interval to check for expired certificates every seven days, the expiration monitor runs on day eight.

Default: Disabled

Next start date:

Specifies the date for the next scheduled check. This allows the deployment manager to be stopped and restarted without resetting the date.

Notifications

Use this page to specify the generic notification definitions that are used in certificate expiration monitors.

To view this administrative console page, complete the following steps:

1. Click **Security > SSL certificate and key management**.
2. Under Configuration settings, click **Manage certificate expiration**.
3. Under Related items, click **Notifications**.

Button	Resulting action
New	Adds a notification. The notification configures how the expiration monitor notifies the administrator of certificates that will expire within the specified threshold.
Delete	Deletes an existing notification.

Notification name:

Specifies the notification name.

Message log:

Specifies that this configuration intends to log certificate expiration information to the message log file.

List of e-mail addresses:

Specifies the e-mail addresses that are sent notifications when certificates fall within the expiration threshold. You must specify the SMTP server for each e-mail address. If an e-mail address is not specified, by default the application server assumes that the SMTP server is ″smtp-server.″ For example, if you type *name*@domain, the SMTP server will be smtp-server.domain.

Notifications settings

Use this page to set properties for new notifications used in certificate expiration monitors.

To view this administrative console page, complete the following steps:

1. Click **Security > SSL certificate and key management**.
2. Under Configuration settings, click **Manage certificate expiration**.
3. Under Related items, click **Notifications > New**.

Notification name:

Specifies the name of the notification configuration.

Data type: Text

Message log:

Specifies that this configuration will log certificate expiration information to a message log file.

Default: Disabled

Email sent to notification list:

Specifies that this configuration intends to send certificate expiration information in an e-mail to the e-mail list.

Default: Disabled

Email address to add:

Specifies the e-mail addresses that are sent notifications when certificates fall within the expiration threshold. You must specify the SMTP server for each e-mail address. If an e-mail address is not specified, by default the application server assumes that the SMTP server is ″smtp-server.″ For example, if you type *name*@domain, the SMTP server will be smtp-server.domain.

Data type: Text (format as valid Internet mail address)

Add:

Adds the e-mail address to the right-hand list.

Remove:

Removes the e-mail address from the right-hand list.

Outgoing mail (SMTP) server:

Specifies the SMTP server to be used with the e-mail address. If none is specified, the e-mail realm will be used.

Key management for cryptographic uses

WebSphere Application Server provides a framework for managing keys (secret keys or key pairs) that applications use to perform cryptographic operations on data. The key management framework provides an application programming interface (API) for retrieving these keys. Keys are managed in keystores so the keystore type can be supported by WebSphere Application Server, provided that the keystores can store the referenced key type. You can configure keys and scope keystores so that they are visible only to particular processes, nodes, clusters, and so on.

The key management infrastructure is based on two key configuration object types: key sets and key set groups. WebSphere Application Server uses a key set to manage instances of keys of the same type. You can configure a key set to generate a single key or a key pair, depending on the key or key pair generator class. A key set group manages one or more key sets and enables you to configure and generate different key types at the same time. For example, if your application needs both a secret key and key pair for cryptographic operations, you can configure two key sets, one for the key pair and one for the secret key that the key set group manages. The key set group controls the auto-generation characteristics of the keys, including the schedule. The framework can automatically generate keys on a scheduled basis, such as on a particular day of the week and time of day, so that key generation is done during off-peak hours.

Figure 1 shows an example of a key set group that is configured to manage two key sets: key set 1 and key set 2.

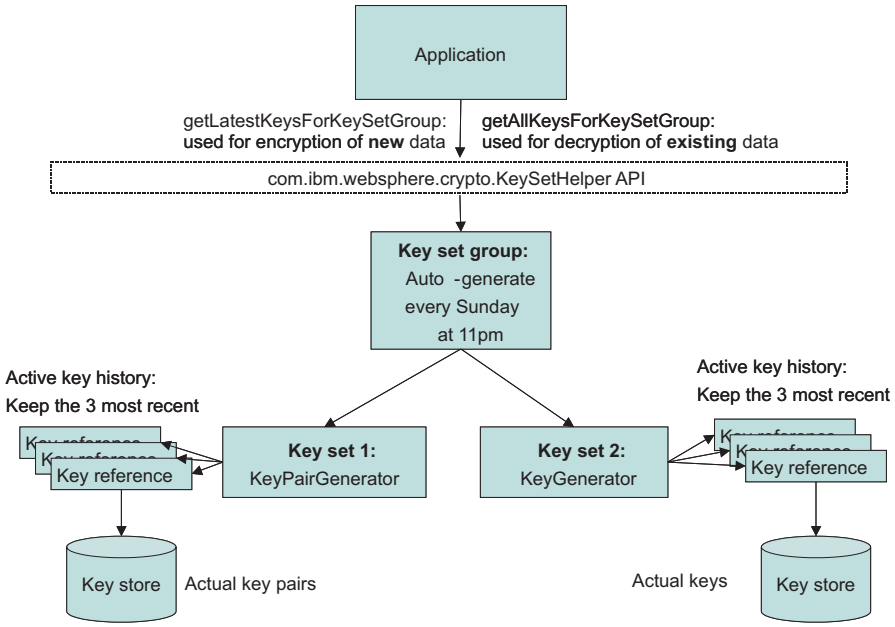


Figure 1: Key Management Infrastructure

Figure 18.

Key set 1 generates key pairs. Key set 2 generates secret keys. The application needs both types of keys for its cryptographic operations, signing and encryption, on data. The keys for each key set need to be

generated in tandem. The application stores the key set group name with the encrypted data. The key set group generates a new set of keys every Sunday night at 11 P.M.. The application maintains key generation data for two weeks.

Creating a key set configuration

You can use key sets to manage multiple instances of cryptographic keys. WebSphere Application Server uses keys to encrypt or sign outbound data, and decrypt or verify inbound data during cryptographic operations.

You must have write-access to the keystore that will contain the keys after you generate them from a key set. However, if you want to generate keys outside of WebSphere Application Server, you can reference the keys from a read-only keystore that contains a secret key that you can access when you generate the keys. If you are creating a key pair using an X509Certificate and a PrivateKey object , see “Example: Developing a key or key pair generation class for automated key generation” on page 1510.

Complete the following steps in the administrative console:

1. Decide whether you want to create the key set at the cell scope or below the cell scope at the node, server, or cluster, for example:
   - To create a key set at the cell scope, click **Security > SSL certificate and key management > Key sets**.
   - To create a key set at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration* > Key sets**.
2. Click **New** to create a new key set.
3. Type a key set name. For example, `CellmyKey`.
4. Type a key alias prefix name. For example, `myKey`. This field specifies the prefix for the key alias when the new key is generated and stored in the keystore. Following the prefix is the key reference version number, for example, 2, so that the full key alias name would be `myKey_2`. If the key reference already has a specified alias for a key that exists in the keystore, then WebSphere Application Server ignores this field.
5. Type a key password. The key password protects the key in the keystore. This password is ignored by WebSphere Application Server if you already specified a password for the key alias reference. To check for a key reference password, click **Active key history** under Additional Properties. The key reference password protects keys that are generated by a key generator class.
6. Type the password again to confirm it.
7. **Optional:** Type the key generator class name. For example, `com.ibm.ws.security.ltpa.LTPAKeyGenerator`. The class name generates keys. If the class implements com.ibm.websphere.crypto.KeyGenerator, then a getKey method returns a java.security.Key object that is set in the keystore using the setKey method without a certificate chain. If the class implements com.ibm.websphere.crypto.KeyPairGenerator, then a getKeyPair method returns a com.ibm.websphere.crypto.KeyPair object that contains either a java.security.PublicKey and java.security.PrivateKey or a java.security.cert.Certificate and a java.security.PrivateKey object. The key generator class and the KeySetHelper API specify the details of the keys that are generated.
8. **Optional:** Select **Delete key references that are beyond the maximum number of keys** if you do not want old keys saved in the keystore after WebSphere Application Server removes their references from the Active key history listing. The Active key history lists the keys that the KeySetHelper API is currently tracking. The number of keys in the list is equal to the number of keys that you specify in **Maximum number of keys referenced**.
9. Type a numeric value for the maximum number of keys referenced. For example, if you type `3` and select **Delete key references that are beyond the maximum number of keys**, the fourth key version generation automatically triggers WebSphere Application Server to delete the first key version from the keystore. If you choose not to delete the old keys, they do not display in the Active key history list but instead remain in the keystore where you can remove them manually.

10. Select a keystore from the drop-down list.
    - Select a JCEKS keystore if you are storing a secret key.
    - Select any keystore if you are storing a key pair with an X509Certificate and PrivateKey object.
11. **Optional:** Select **Generates key pair** if your key generator class name implements the com.ibm.websphere.crypto.KeyPairGenerator interface instead of the com.ibm.websphere.crypto.KeyGenerator interface. This option designates that the key references a key pair instead of a single key. A key pair contains both a public key and a private key. The WebSphere Application Server run time determines whether or not key pairs are stored and loaded differently than single keys.
12. **Optional:** Click **Apply** if you want to select **Active key history** under Additional Properties to add alias references or generate more keys.
    a. Click **Active key history**.
    b. Click **Add key alias reference** if you are not using the key generator class name to add key alias references to the keys that already exist in the keystore. Use this option to retrieve the keys from a read-only keystore without the key set generating them.
    c. Type an alias reference.
    d. Click **Generate key** if you want to generate a key using the class name that you defined in the key sets panel. Each new key increments numerically, for example, `myAlias_2`.
    e. Click **Apply**.
13. Click the key set name in the navigation path at the top of the panel.
14. Click **OK** and **Save**.

You have created a key set that you can manage using the **Active key history** link. You can generate keys manually to associate them with specified key sets.

After you generate new keys from a key set, you can access them programmatically using the com.ibm.websphere.crypto.KeySetHelper API. You must have Java 2 Security permissions, if enabled, to access keys in key sets. Specify the key set name within the fine-grained permissions, as in the following code sample: `WebSphereRuntimePermission "getKeySets.`*`keySetName`*`"`. For more information, see “Example: Retrieving the generated keys from a key set group” on page 1508. To generate multiple key types at the same time or to schedule the key generation on a specific schedule, see “Creating a key set group configuration” on page 1507.

Active key history collection

Use this page to manage key alias references.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration_name***. Under Related items, click **Key Sets > *key set* > Active key history**.

Button	Resulting action
Add key alias reference	Adds a reference to a key that already exists in a key store. If a key generation class is configured, the references are added automatically during generation and do not need to be added manually.
Delete	Deletes an existing key reference. This action does not delete the key in the keystore.
Generate key	Generates a key. The button is displayed only if a generator class name is specified for the key set, and the selected key store is editable.

Key alias reference:

Specifies the name of the alias as it appears in the keystore.

Add key alias reference settings

Use this page to access key alias reference information.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration_name***. Under Related items, click **Key Sets > *key set* > Active key history > Add key alias reference**.

Alias reference:

Specifies the name of the alias as it appears in the key store.

Data type: Text

Password:

Specifies the key password to get access to the key. This password is enforced by the keystore for that specific key. If the key does not have a password, this field can be left blank.

Data type: Text

Confirm password:

Confirms the password entered in the previous field.

Data type: Text

Key sets collection

Use this page to manage key sets, which control a set of key instances of the same type for use in cryptographic operations. The keys can either be generated using a custom class or reference keys that already exist in a keystore.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key sets**.

Button	Resulting action
New	Adds a new key set.
Delete	Deletes an existing key set. Make sure the key set is not referenced by a key set group before deleting it.

Key set name:

Specifies the key set name that is used to select the key set from a key set group and from runtime application programming interfaces (API).

Key store:

Specifies the key store that contains the keys for storage, retrieval, or both.

Key alias prefix name:

Specifies the prefix for the key alias when a new key is generated and stored in a key store. The rest of the key alias comes from the key reference version number.

For example, if the alias prefix is mykey and the key reference version is 2, the keystore references the key using alias mykey_2. If the key reference already has a specified alias for a key already existing in the keystore, this field is ignored.

Key sets settings

Use this page to set the properties for a new key set.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key sets > New**.

Key set name:

Specifies the key set name that is used to select the key set from a key set group and from runtime application programming interfaces (API).

Data type: Text

Key alias prefix name:

Specifies the prefix for the key alias when a new key is generated and stored in a keystore. The rest of the key alias comes from the key reference version number. For example, if the alias prefix is mykey and the key reference version is 2, the keystore references the key using alias mykey_2. If the key reference already has a specified alias for a key already existing in the keystore, this field is ignored.

Data type: Text

Key password:

Specifies the password used to protect the key in the keystore. If a password is specified in the key reference as well, this password is ignored. This password is used for keys that get generated by a key generator class.

Data type: Text

Confirm password:

Specifies the same password again to confirm it was entered correctly the first time.

Data type: Text

Key generator class name:

Specifies the class name that generates keys. If the class implements com.ibm.websphere.crypto.KeyGenerator, then a getKey() method should return a java.security.Key object that is set in the key store using the setKey method without a certificate chain. The key store type associated with the key set must support storing keys without certificates, such as JCEKS.

Data type: Text

If the class implements com.ibm.websphere.crypto.KeyPairGenerator, then a getKeyPair() method should return a com.ibm.websphere.crypto.KeyPair object containing either a java.security.PublicKey and java.security.PrivateKey, or a java.security.cert.Certificate[] and a java.security.PrivateKey. The key

generator class and the caller of the KeySetHelper API should know the details of the keys that are generated. This framework does not need to understand the key algorithms and lengths.

Delete key references that are beyond the maximum number of keys:

Specifies that the keys are deleted from the keystore at the same time that the key reference is deleted. The server deletes the older key references as the Maximum number of keys referenced value is exceeded.

Maximum number of keys referenced:

Specifies the maximum number of key instances that are returned when keys from this key set are requested. The oldest key reference gets removed whenever a new key reference gets generated after the maximum has been reached.

Data type: Integer
Default: 3

Key store:

Specifies the key store that contains the keys for storage, retrieval, or both.

Data type: Text

Generates key pair:

Specifies that a key references a key pair instead of a key. The key pair contains both a public key and a private key.

Creating a key set group configuration

A key set group manages one or more key sets. WebSphere Application Server uses key set groups to automatically generate cryptographic keys or multiple synchronized key sets.

Complete the following steps in the administrative console:

1. Decide whether you want to create the key set group at the cell scope or below the cell scope at the node, server, or cluster, for example.
   - To create a key set group at the cell scope, click **Security > SSL certificate and key management > Key set groups**.
   - To create a key set group at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *SSL_configuration* > Key set groups**.
2. You can choose to generate a key for an existing key set group, delete an existing key set group, or create a new key set group.
   - To generate a key for an existing key set group, select a key set group from the list of existing key set groups, and click **Generate keys**. You have generated a new key for each key set in the selected group.
   - To delete an existing key set group, select a key set group from the list of existing key set groups, and click **Delete**. You have deleted the key set group.
   - To create a new key set group, go to step 3.

   CAUTION:
   Do not delete the cell or node LTPAKeySetGroup, which is used by the Lightweight Third Party Authentication (LPTA) mechanism.

3. Click **New** to create a new key set group.
4. Type a key set group name. You can reference this name by using the com.ibm.websphere.crypto.KeySetHelper API to retrieve the managed keys from an application.
5. Select one or more key sets from the Key sets list.

   Note: If the key set(s) you want is not listed, make sure that it was created at the same scope or a higher scope than where you are creating the new key set group.
6. Click **Add** to add the selected key set(s) to the new key set group.
7. Select **Automatically generate keys** to generate the new keys on a schedule. If you decide to generate keys automatically, then you must specify a scheduled time of day.
8. Specify the scheduled time to generate keys automatically in hours and minutes, A.M. or P.M., or every 24 hours.
9. You can choose to generate new keys on a specific day or at an interval.
   - Select **Generate on a specific day**. Select a day of the week from the drop-down list, and type a repeat interval number for the number of days between each key generation. This choice enables you to schedule key generation when your systems are least busy.
   - Select **Generate at an interval**. Type a repeat interval number for the number of days between each key generation. This choice enables you to schedule key generation more frequently than once a week.

   Note: The **Next start date** is a read-only field that specifies the date for the next scheduled generation. You can stop and restart the deployment manager or base application server without resetting this date. If you do not see the next start date appear after changing the configuration, click **OK** to save it, then check that the next start date displays.
10. Click **Save**.

You have created a new key set group to manage key sets and key generation on a schedule.

After you generate new keys from a key set, you can access them programmatically using the com.ibm.websphere.crypto.KeySetHelper API. You must have Java 2 Security permissions, if enabled, to access keys in key sets. Specify the key set name within the fine-grained permissions, as in the following code sample: `WebSphereRuntimePermission "getKeySets.keySetName"`. For more information, see "Example: Retrieving the generated keys from a key set group."

Example: Retrieving the generated keys from a key set group

This example shows how applications can use the com.ibm.websphere.crypto.KeySetHelper API to retrieve managed keys from the KeySet or KeySetGroup configurations. Use the com.ibm.websphere.crypto.KeySetHelper API to get either the latest set of keys or the all keys in the KeySet or KeySetGroup object.

Use the latest keys when performing any new cryptographic operations. All of the other keys that are defined in the KeySet or KeySetGroup object are for the validation of previously performed cryptographic operations.

The following example uses a method that an application might use to initialize the keys in the associated KeySetGroup object. The application might want to store the keys in two separate maps, one for generation and one for validation. Refer to the API documentation for KeySetHelper API to determine which Java 2 Security requirements are required.

```
/**
     * Initializes the primary and secondary Maps used for initializing the keys.
     */

    public void initializeKeySetGroupKeys() throws com.ibm.websphere.crypto.KeyException
    {
        java.util.Map generationKeys = null;
```

```
    java.util.Map validationKeys = null;

    PublicKey tempPublicKey = null;
    PrivateKey tempPrivateKey = null;
    byte[] tempSharedKey = null;

    keySetGroupName = "ApplicationKeySetGroup";
    com.ibm.websphere.crypto.KeySetHelper ksh = com.ibm.websphere.crypto.KeySetHelper.getInstance();
    generationKeys = ksh.getLatestKeysForKeySetGroup(keySetGroupName);

    /***
    *  Latest keys: {
    *  KeyPair_3=com.ibm.websphere.crypto.KeyPair@64ec64ec,
    *  Secret_3=javax.crypto.spec.SecretKeySpec@fffe8aa7
    *  }
    ***/

    if (generationKeys != null)
    {
        Iterator iKeySet = generationKeys.keySet().iterator();

        while (iKeySet.hasNext())
        {
            String keyAlias = (String)iKeySet.next();

            Object key = generationKeys.get(keyAlias);

            if (key instanceof java.security.Key)
            {
                tempSharedKey = ((java.security.Key)key).getEncoded();
            }
            else if (key instanceof com.ibm.websphere.crypto.KeyPair)
            {
                java.security.Key publicKeyAsSecret =
((com.ibm.websphere.crypto.KeyPair)key).getPublicKey();
                tempPublicKey = new PublicKey(publicKeyAsSecret.getEncoded());
                java.security.Key privateKeyAsSecret =
((com.ibm.websphere.crypto.KeyPair)key).getPrivateKey();
                tempPrivateKey = new PrivateKey(privateKeyAsSecret.getEncoded());
            }
        }

        // save these for use later, if necessary
        validationKeys = ksh.getAllKeysForKeySetGroup(keySetGroupName);

        /***
        *  All keys: {
        *  version_1=
        *      {Secret_1=javax.crypto.spec.SecretKeySpec@178cf,
        *       KeyPair_1=com.ibm.websphere.crypto.KeyPair@1c121c12},
        *  version_2=
        *      {Secret_2=javax.crypto.spec.SecretKeySpec@17a77,
        *       KeyPair_2=com.ibm.websphere.crypto.KeyPair@182e182e},
        *  version_3=
        *      {Secret_3=javax.crypto.spec.SecretKeySpec@fffe8aa7,
        *       KeyPair_3=com.ibm.websphere.crypto.KeyPair@4da04da0}
        *  }
        ***/
    }
    else
    {
        throw new com.ibm.websphere.crypto.KeyException("Could not generateKeys.");
    }
}
```

Example: Developing a key or key pair generation class for automated key generation

A class that generates keys for cryptographic operations can be created automatically. With this capability, the key management infrastructure can maintain a list of keys for a predefined key set, and applications can access these keys.

You can schedule new key generation at predefined frequencies. Remember that key generation frequency affects the security of your data. For example, for persistent data, you might schedule key generation less frequently than for real time communications, which require that the keys be generated more often as old keys expire.

When you develop a key generation class, decide if you are creating a shared key or a key pair because this decision determines the interface you must use.

If you are developing shared keys, refer to the following example, which uses the KeyGenerator class to implement the com.ibm.websphere.crypto.KeyGenerator interface. The interface returns a java.security.Key key, which is stored as a SecretKey in a JCEKS keystore type. You can use any other keystore type that supports storing secret keys.

```
package com.ibm.test;

import java.util.*;
import com.ibm.ws.ssl.core.*;
import com.ibm.ws.ssl.config.*;
import com.ibm.websphere.crypto.KeyException;

public class KeyGenerator implements com.ibm.websphere.crypto.KeyGenerator
{
    private java.util.Properties customProperties = null;
    private java.security.Key secretKey = null;

    public KeyGenerator()
    {
    }

     /**
      * <p>
      * This method is called to pass any custom properties configured with
      * the KeySet to the implementation of this interface.
      * </p>
      *
      * @param java.util.Properties
      **/
    public void init (java.util.Properties customProps)
    {
        customProperties = customProps;
    }

     /**
      * <p>
      * This method is called whenever a key needs to be generated either
      * from the schedule or manually requested. The key is stored in the
      * KeyStore referenced by the configured KeySet that contains the
      * keyGenerationClass implementing this interface. The implementation of
      * this interface manages the key type. The user of the KeySet
      * must know the type that is returned by this keyGenerationClass.
      * </p>
      *
      * @return java.security.Key
      * @throws com.ibm.websphere.crypto.KeyException
      **/
    public java.security.Key generateKey () throws KeyException
    {
        try
        {
            // Assume generate3DESKey is there to create the key.
            byte[] tripleDESKey = generate3DESKey();
```

```
            secretKey = new javax.crypto.spec.SecretKeySpec(tripleDESKey, 0, 24, "3DES");

            if (secretKey != null)
            {
                return secretKey;
            }
            else
            {
                throw new com.ibm.websphere.crypto.KeyException ("Key could not be generated.");
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();  // handle exception
        }
    }
}
```

If you are developing a key pair, refer to the following example, which uses the KeyPairGenerator class to implement the com.ibm.websphere.crypto.KeyPairGenerator interface.

```
package com.ibm.test;

import java.util.*;
import javax.crypto.spec.SecretKeySpec;
import com.ibm.websphere.crypto.KeyException;

/**
 * <p>
 * This implementation defines the method to generate a java.security.KeyPair.
 * When a keyGeneration class implements this method, the generateKeyPair method
 * is called and a KeyPair is stored in the keystore. The isKeyPair
 * attribute is ignored since the KeyGenerationClass is an
 * implementation of KeyPairGenerator. The isKeyPair attributes is for when
 * the keys already exist in a KeyStore, and are just read (not
 * generating them).
 * </p>
 * @author IBM Corporation
 * @version WebSphere Application Server 6.1
 * @since WebSphere Application Server 6.1
 **/
public class KeyPairGenerator implements com.ibm.websphere.crypto.KeyPairGenerator
{
    private java.util.Properties customProperties = null;

    public KeyPairGenerator()
    {
    }

     /**
      * <p>
      * This method is called to pass any custom properties configured with
      * the KeySet to the implementation of this interface.
      * </p>
      *
      * @param java.util.Properties
      **/
    public void init (java.util.Properties customProps)
    {
        customProperties = customProps;
    }

     /**
      * <p>
      * This method is called whenever a key needs to be generated either
      * from the schedule or manually requested and isKeyPair=true in the KeySet
      * configuration. The key is stored in the KeyStore referenced by
      * the configured KeySet which contains the keyGenerationClass implementing
      * this interface. The implementation of this interface manages the
      * type of the key. The user of the KeySet must know the type that
      * is returned by this keyGenerationClass.
```

```
     * </p>
     *
     * @return com.ibm.websphere.crypto.KeyPair
     * @throws com.ibm.websphere.crypto.KeyException
     **/
    public com.ibm.websphere.crypto.KeyPair generateKeyPair () throws KeyException
    {
        try
        {
            java.security.KeyPair keyPair = generateKeyPair();

            // Store as SecretKeySpec
            if (keyPair != null)
            {
                java.security.PrivateKey privKey = keyPair.getPrivate();
                java.security.PublicKey pubKey = keyPair.getPublic();

                SecretKeySpec publicKeyAsSecretKey = new SecretKeySpec
                    (pubKey.getEncoded(), "RSA_PUBLIC");
                SecretKeySpec privateKeyAsSecretKey = new SecretKeySpec
                    (privKey.getEncoded(), "RSA_PRIVATE");

                com.ibm.websphere.crypto.KeyPair pair = new
    com.ibm.websphere.crypto.KeyPair(publicKeyAsSecretKey, privateKeyAsSecretKey);
                return pair;
            }
            else
            {
                throw new com.ibm.websphere.crypto.KeyException ("Key pair could
                not be generated.");
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();  // handle exception
        }
    }
}
```

This interface returns a com.ibm.websphere.crypto.KeyPair key pair, which can contain either a X509Certificate and PrivateKey object or PublicKey and PrivateKey objects. If the com.ibm.websphere.crypto.KeyPair interface contains aX509Certificate and PrivateKey object, the certificate and private key are stored in the keystore. Consequently, they can use any KeyStore type.

If the com.ibm.websphere.crypto.KeyPair interface contains PublicKey and PrivateKey objects, you must convert the encoded values to the SecretKeySpec object in order to store them. The WebSphere Application Server runtime stores and retrieves the key pair as secret keys. The runtime converts the key pair back to PublicKey and PrivateKey objects when the server retrieves the pair during the handshake.

Use the following constructors to develop the com.ibm.websphere.crypto.KeyPair interface:

- Public and private constructor

  ```
  public KeyPair(java.security.Key publicKey, java.security.Key privateKey)
  ```

- Certificate and private constructor.

  ```
  public KeyPair(java.security.cert.Certificate[] certChain,
  java.security.Key privateKey)
  ```

The previous example code shows the KeyPairGenerator class using the public and private constructor. Each call to this class generates a new and unique key pair, and this class is invoked by a KeySet to create a new key pair when `isKeyPair=true`. The version number in the key set increments each time it is called.

Key set groups collection

Use this page to manage groups of public, private, and shared keys. These key groups enable the application server to control multiple sets of Lightweight Third Party Authentication (LTPA) keys.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key set groups**.

Button	Resulting action
New	Adds a key set group. A key set group combines one or more key sets together as a single key set group. It allows the generation of multiple different types of keys to occur at the same time. A single key set represents one type of key, so a key set group allows you to group the different types.
Delete	Deletes an existing key set group. You must be sure that there are no other references to this key set group before you delete it.
Generate keys	Generates keys for key set group. The system generates keys for each key set within the key set group so that the keys remain synchronized with each other in terms of version. You must configure a valid key generation class and a key store that is writable. See the com.ibm.websphere.crypto.KeySetHelper application programming interfaces (APIs) to enable the use of keys that are managed by a KeySetGroup or KeySet.

Key set group name:

Specifies the name of the key set group used to reference it.

Automatically generate keys:

Specifies that the keys are to be generated automatically on a schedule.

Key set groups settings

Use this page to create new key set groups.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under Related items, click **Key set groups > New**.

Key set group name:

Specifies the name of key set group used. This name can be referenced using the com.ibm.websphere.crypto.KeySetHelper API to retrieve the managed keys from an application.

Data type: Text

Key sets:

Specifies a set of key instances of the same type for use in cryptographic operations.

Add:

Specifies to add the selected key set part of this key set group.

Remove:

Specifies to remove the selection from the **Key sets** list.

Automatically generate keys:

Specifies that the keys are generated automatically on a schedule. When a new key is generated, the `security.xml` is updated and saved by the runtime to track the key reference version. This can cause save conflicts when updating the same file from admin applications.

Default: Disabled

Scheduled time for generation:

Specifies the scheduled time when the system generates selected key set group or groups. You can specify the scheduled time in hours and minutes; specify either A.M. or P.M., or specify 24-hour. You can also specify the day of the week you want the scheduled event to occur. It is recommended that you set this event to occur during a low peak time, especially for keys that are used by runtime for token validation.

Data type Integer
Default: 0, 0
Range: 1–12, 0–59

Generate on a specific day:

Specifies whether to have the generation occur on a specific day of the week. It is best to auto-generate keys during a low peak day.

Default: Enabled

Weekday:

Specifies the day of the week on which the expiration monitor will run if the Check on a specific day option is selected.

Default: Sunday
Range: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Repeat interval:

Specifies the period of time between each schedule time to check for expired certificates or the interval between schedule checks.

Default: Daily
Range: Daily, Weekly

Generate at an interval:

Specifies to generate keys at the specified frequency regardless of the day of the week on which generation occurs.

Default: Disabled

Next start date:

Specifies the date for the next scheduled check. This allows the deployment manager to be stopped and restarted without resetting the date.

Configuring security with scripting

You can configure security with scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

If you enable security for a WebSphere Application Server cell, supply authentication information to communicate with servers.

The `sas.client.props` and the `soap.client.props` files are located in the properties directory for each WebSphere Application Server profile, *profilePath*`/properties`.

- The nature of the properties file updates required for running in secure mode depend on whether you connect with a Remote Method Invocation (RMI) connector, or a SOAP connector:
  - If you use a Remote Method Invocation (RMI) connector, set the following properties in the `sas.client.props` file with the appropriate values:

    ```
    com.ibm.CORBA.loginUserid=
    com.ibm.CORBA.loginPassword=
    ```

    Also, set the following property:

    ```
    com.ibm.CORBA.loginSource=properties
    ```

    The default value for this property is `prompt` in the `sas.client.props` file. If you leave the default value, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang.
  - If you use a SOAP connector, set the following properties in the `soap.client.props` file with the appropriate values:

    ```
    com.ibm.SOAP.securityEnabled=true
    com.ibm.SOAP.loginUserid=
    com.ibm.SOAP.loginPassword=
    ```

    Optionally, set the following property:

    ```
    com.ibm.SOAP.loginSource=none
    ```

    The default value for this property is `prompt` in the `soap.client.props` file. If you leave the default value, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang.
- To specify user and password information, choose one of the following methods:
  - Specify user name and password on a command line, using the **-user** and **-password** commands. For example:

    ```
    wsadmin -conntype RMI -port 2809 -user u1 -password secret1
    ```
  - Specify user name and password in the `sas.client.props` file for a RMI connector or the `soap.client.props` file for a SOAP connector.

  If you specify user and password information on a command line and in the `sas.client.props` file or the `soap.client.props` file, the command line information overrides the information in the `props` file.

  Note: On UNIX system, the use of -password option may result in security exposure as the password information becomes visible to the system status program such as ps command which can be invoked by other user to display all the running processes. Do not use this option if security exposure is a concern. Instead, specify user and password information in the soap.client.props file for SOAP connector or sas.client.props file for RMI connector. The soap.client.props and sas.client.props files are located in the properties directory of your WebSphere Application Server profile.

Enabling and disabling administrative security using scripting

You can use scripting to enable or disable administrative security.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

The default profile sets up procedures so that you can enable and disable administrative security based on LocalOS registry.

- To determine if application security is enabled or disabled by looking at the value of the appEnabled field in the WCCM security model, use the **isAppEnabled** command, for example:
  - Using Jacl:

    ```
    $AdminTask isAppSecurityEnabled {-interactive}
    ```
  - Using Jython:

    ```
    AdminTask.isAppSecurityEnabled ('[-interactive]')
    ```

  This command returns a value of `true` if appEnabled is set to `true`. Otherwise, returns a value of `false`.
- To determine if administrative security is enabled or disabled by looking at the value of the enabled field in the WCCM security model, use the **isGlobalSecurity** command, for example:
  - Using Jacl:

    ```
    $AdminTask isGlobalSecurity {-interactive}
    ```
  - Using Jython:

    ```
    AdminTask.isGlobalSecurity ('[-interactive]')
    ```

  Returns a value of `true` if enabled is set to `true`. Otherwise, returns a value of `false`.
- To set administrative security based on the passed in value, use the **setGlobalSecurity** command. For example:
  - Using Jacl:

    ```
    $AdminTask setGlobalSecurity {-interactive}
    ```
  - Using Jython:

    ```
    AdminTask.setGlobalSecurity ('[-interactive]')
    ```

  Returns a value of `true` if the enabled field in the WCCM security model is successfully updated. Otherwise, returns a value of `false`.
- You can use the **help** command to find out the arguments that you need to provide with this call, for example:
  - Using Jacl:

    ```
    securityon help
    ```

    Example output:

    ```
    Syntax: securityon user password
    ```
  - Using Jython:

    ```
    securityon()
    ```

    Example output:

    ```
    Syntax: securityon(user, password)
    ```
- To enable administrative security based on the LocalOS registry, use the following procedure call and arguments:
  - Using Jacl:

    ```
    securityon user1 password1
    ```
  - Using Jython:

    ```
    securityon('user1', 'password1')
    ```
- To disable administrative security based on the LocalOS registry, use the following procedure call:

– Using Jacl:

```
securityoff
```

– Using Jython:

```
securityoff()
```

Enabling and disabling LTPA authentication

There are sample scripts located in the <WAS_ROOT>/bin directory on how to enable and disable LTPA authentication. The scripts are:

– LTPA_LDAPSecurityProcs.py (python script)

– LTPA_LDAPSecurityProcs.jacl (jacl script)

Note: The scripts hard code the type of LDAP server and base distinguished name (baseDN). The LDAP server type is hardcoded as IBM_DIRECTORY_SERVER and the baseDN is hardcoded as o=ibm,cn=us.

Enabling and disabling Java 2 security using scripting

You can enable or disable Java 2 security with scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to enable or disable Java 2 security:

1. Identify the security configuration object and assign it to the security variable:
   - Using Jacl:

     ```
     set security [$AdminConfig list Security]
     ```

   - Using Jython:

     ```
     security = AdminConfig.list('Security')
     print security
     ```

   Example output:

   ```
   (cells/mycell|security.xml#Security_1)
   ```

2. Modify the enforceJava2Security attribute to enable or disable Java 2 security. For example:
   - To enable Java 2 security:
     – Using Jacl:

       ```
       $AdminConfig modify $security {{enforceJava2Security true}}
       ```

     – Using Jython:

       ```
       AdminConfig.modify(security, [['enforceJava2Security', 'true']])
       ```

   - To disable Java 2 security:
     – Using Jacl:

       ```
       $AdminConfig modify $security {{enforceJava2Security false}}
       ```

     – Using Jython:

       ```
       AdminConfig.modify(security, [['enforceJava2Security', 'false']])
       ```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

Enabling authentication in the file transfer service using scripting

You can enable authentication in the file transfer service using scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

In WebSphere Application Server Network Deployment, V5.0.1 or later, the file transfer service is enhanced to provide role-based authentication. Two versions of the file transfer Web application are provided. By default, the version that does not authenticate its caller is installed. This default supports compatibility between the WebSphere Application Server Network Deployment, V5.0 and V5.0.1 or later.

Turning the file transfer authentication on is recommended to prevent unauthorized use of the file transfer application; however, if you have any V5.0 clients in your Network Deployment environment, they cannot communicate with the secured file transfer application if global security is turned on.

In WebSphere Application Server V6.x, mixed cells are supported and file transfer has become a system application. If all of the nodes in the cell are of V5.0.1 or later, you can activate authentication in the file transfer service by redeploying the file transfer application at the deployment manager. The compatible version is shipped in the *app_server_root*/systemApps/filetransfer.ear directory. The secured version is provided in the *app_server_root*/systemApps/filetransferSecured.ear directory.

- A wsadmin Jacl script is provided to help you redeploy the file transfer. The script is called redeployFileTransfer.jacl and is located in the *app_server_root*/bin directory. After the deployment manager and all the nodes are upgraded to WebSphere Application Server Network Deployment, version 5.0.1 or later, you can deploy the secured file transfer service by running the script. The syntax for running the script from the bin directory is the following:
  –
  ```
  wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationXxx cellName nodeName serverName"
  ```
  where Xxx is **On** or **Off**.
  For Windows systems, use wsadmin or wsadmin.bat. For Linux and UNIX systems, use wsadmin.sh. For OS/400 systems, use wsadmin.
  – For example, when running the script to enable use of the filetransferSecured.ear file, the syntax is similar to the following example:
  ```
  wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOn managedCell managedCellManager dmgr"
  ```
  or
  ```
  wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOn baseCell base server1"
  ```
- If you want to go return to running the file transfer service without authentication, you can run the script as shown in the following example:
  ```
  wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOff baseNodeCell baseNode server1"
  ```
  or
  ```
  wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOff managedCell managedCellManager dmgr"
  ```

You must restart the server for the change to take affect.

Propagating security policy of installed applications to a JACC provider using wsadmin scripting

It is possible that you have applications installed prior to enabling the Java Authorization Contract for Containers (JACC)-based authorization. You can start with default authorization and then move to an external provider-based authorization using JACC later.

Also, during application installation or modification you might have had problems propagating the security policy information to the JACC provider. For example, network problems might occur, the JACC provider might not be available, and so on. For these cases, the security policy of the previously installed applications does not exist in the JACC provider to make the access decisions. One choice is to reinstall

the applications involved. However, you can avoid reinstalling by using the wsadmin scripting tool. Use this tool to propagate information to the JACC provider independent of the application installation process. The tool eliminates the need for reinstalling the applications.

The tool uses the SecurityAdmin MBean to propagate the policy information in the deployment descriptor of any installed application to the JACC provider. You can invoke this tool using wsadmin at the base application server for base and deployment manager level for Network Deployment. Note that the SecurityAdmin MBean is available only when the server is running.

Use `propagatePolicyToJACCProvider(String appNames)` to propagate the policy information in the deployment descriptor of the enterprise archive (EAR) files to the JACC provider. If the RoleConfigurationFactory and the RoleConfiguration interfaces are implemented by the JACC provider, the authorization table information in the binding file of the EAR files is also propagated to the provider. See the *Securing applications and their environment* PDF for more information about these interfaces.

The `appNames String` contains the list of application names, delimited by a colon (:), whose policy information must be stored in the provider. If a null value is passed, the policy information of the deployed applications is propagated to the provider.

Also, be aware of the following items:

- Before migrating applications to the Tivoli Access Manager JACC provider, create or import the users and groups that are in the applications to Tivoli Access Manager.
- Depending on the application or the number of applications that are propagated, you might have to increase the request time-out period either in the `soap.client.props` file in the directory *profile_root*`/properties` (if using SOAP) or in the `sas.client.props` file (if using RMI) for the command to complete. You can set the request time-out value to 0 to avoid the timeout problem, and change it back to the original value after the command is run.

1. Configure your JACC provider in WebSphere Application Server.

   See the *Securing applications and their environment* PDF for more information.
2. Restart the server.
3. Enter the following commands:

```
//use the SecurityAdmin MBean at the Deployment Manager or the unmanaged base
//application server connect to the appropriate process (Deployment Manager or
//base application server)
wsadmin -user serverID -password serverPWD
// To get the SecurityAdmin MBean for Deployment Manager
wsadmin> set secadm [$AdminControl queryNames type=SecurityAdmin,process=dmgr,*]
// or to get the SecurityAdmin MBean for a unmanaged base application server
//(replace the process name to match your configuration)
wsadmin> set secadm [$AdminControl queryNames
         type=SecurityAdmin,process=server1,*]
// to propagate specific applications security policy information
wsadmin>set appNames [list app1:app2]
// or to propagate all applications installed
wsadmin>set appNames [list null]

// Run the command to propagate
wsadmin>$AdminControl invoke $secadm propagatePolicyToJACCProvider $appNames
```

Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility

You can use the wsadmin utility to configure Tivoli Access Manager security for WebSphere Application Server.

1. Start WebSphere Application Server.
2. Start the **wsadmin** command-line utility.

Run the **wsadmin** command from the *app_server_root*/bin directory.

3. At the **wsadmin** prompt, enter the following command:

   $AdminTask configureTAM -interactive

   You are prompted to enter the following information:

Option	Description
WebSphere Application Server node name	Specify a single node.
Tivoli Access Manager Policy Server	Enter the name of the Tivoli Access Manager policy server and the connection port. Use the format, *policy_server* **:** *port*. The policy server communication port is set at the time of Tivoli Access Manager configuration. The default port is 7135.
Tivoli Access Manager Authorization Server	Enter the name of the Tivoli Access Manager authorization server. Use the format *auth_server* **:** *port* **:** *priority*. The authorization server communication port is set at the time of Tivoli Access Manager configuration. The default port is 7136. More than one authorization server can be specified by separating the entries with commas. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example: auth_server1**:**7136**:**1,auth_server2**:**7137**:**2. A priority of 1 is still required when configuring against a single authorization server.
WebSphere Application Server administrator's distinguished name	Enter the full distinguished name of the WebSphere Application Server security administrator ID, as created in the ″Creating the security administrative user″ topic in the *Securing applications and their environment* PDF. For example: cn=wasadmin,o=*organization*,c=*country*
Tivoli Access Manager user registry distinguished name suffix	For example: o=*organization*,c=*country*
Tivoli Access Manager administrator's user name	Enter the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This ID is usually, sec_master.
Tivoli Access Manager administrator's user password	Enter the password for the Tivoli Access Manager administrator.
Tivoli Access Manager security domain	Enter the name of the Tivoli Access Manager security domain that is used to store users and groups. If a security domain is not already established at the time of Tivoli Access Manager configuration, click **Return** to accept the default.
Embedded Tivoli Access Manager listening port set	WebSphere Application Server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine so a list of ports is required for the processes. Enter the ports that are used as listening ports by Tivoli Access Manager clients, separated by a comma. If you specify a range of ports, separate the lower and higher values by a colon. For example, 7999, 9990:9999.

Option	Description
Defer	Set to *yes*, this option defers the configuration of the management server until the next restart. Set to *no*, configuration of the management server occurs immediately. Managed servers are configured on their next restart.

4. When all information is entered, select **F** to save the configuration properties or **C** to cancel from the configuration process and discard entered information.

Now enable the JACC provider for Tivoli Access Manager- Enabling the JACC provider for Tivoli Access Manager topic in the *Securing applications and their environment* PDF.

Disabling embedded Tivoli Access Manager client using wsadmin

Follow these steps to unconfigure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager.

1. Start the wsadmin command-line utility. The wsadmin command is found in the*install_dir*/bin directory
2. From the **wsadmin** prompt, enter the following command:

   ```
   WSADMIN>$AdminTask unconfigureTAM -interactive
   ```

   You are prompted to enter the following information:

Option	Description
WebSphere Application Server node name	Enter an asterisk (*) to select all nodes.
Tivoli Access Manager administrator user name	Enter the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This name is usually, `sec_master`.
Tivoli Access Manager administrator user password	Enter the password for the Tivoli Access Manager administrator.
Force	Enter *yes,* if you want to ignore errors when unconfiguring the JACC provider for Tivoli Access Manager. Enter this option as *yes* only when the Tivoli Access Manager domain is in an irreparable state.
Defer	Enter `no`, to force the unconfiguration of the connected server. Enter `No` for the unconfiguration to proceed correctly.

3. When all information is entered, enter `F` to save the properties or `C` to cancel from the unconfiguration process and discard the entered information.
4. Restart all WebSphere Application Server instances for the changes to take effect.

Creating an SSL configuration at the node scope using scripting

An Secure Socket Layer (SSL) configuration references many other configuration objects. To help you make valid selections for the new SSL configuration before you create it, view information about existing configuration objects. Information about existing objects is also useful when you create a node scoped SSL configuration using the **createSSLConfig** command of the AdminTask object.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

To use the information in this task effectively, familiarize yourself with the instructions in the Creating a Secure Sockets Layer configuration topic in the *Securing applications and their environment* PDF. Perform the following task to create an Secure Socket Layer (SSL) configuration at the node scope:

1. List the existing configuration objects. Perform any of the following:
   - List some of the configuration objects that you may need when you create a new SSL configuration.

     For example, you want to see which management scopes have already been defined. If the one you need does not exist you will need to create it.

     – Using Jacl:

     ```
     $AdminTask listManagementScopes {-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02}
     ```

     – Using Jython:

     ```
     AdminTask.listManagementScopes ('[-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02]')
     ```

     This shows an existing cell scope and existing node scope that you can use. If you want to create a different scope, use the **createManagementScope** command of the AdminTask object to define a different one. The valid scope parameters are cell, nodegroup, node, server, cluster, and endpoint. See the Central management of Secure Sockets Layer configurations topic in the *Securing applications and their environment* PDF for more information on scope definitions.
   - List the key stores that exist in the configuration including key stores and trust stores.

     – Using Jacl:

     ```
     $AdminTask listKeyStores
     ```

     – Using Jython:

     ```
     AdminTask.listKeyStores()
     ```

     Example output:

     ```
     CellDefaultKeyStore(cells/BIRKT40Cell02|security.xml#KeyStore_1)
     CellDefaultTrustStore(cells/BIRKT40Cell02|security.xml#KeyStore_2)
     CellLTPAKeys(cells/BIRKT40Cell02|security.xml#KeyStore_3)
     ```

     The previous example only lists the key stores for the default management scope which is also known as the cell scope. To obtain key stores for other scopes, specify the `scopeName` parameter, for example:

     – Using Jacl:

     ```
     $AdminTask listKeyStores {-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 }
     ```

     – Using Jython:

     ```
     $AdminTask listKeyStores ('[-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02]')
     ```

     Example output:

     ```
     CellDefaultKeyStore(cells/BIRKT40Cell02|security.xml#KeyStore_1)
     CellDefaultTrustStore(cells/BIRKT40Cell02|security.xml#KeyStore_2)
     CellLTPAKeys(cells/BIRKT40Cell02|security.xml#KeyStore_3)
     NodeDefaultKeyStore(cells/BIRKT40Cell02|security.xml#KeyStore_1134610924357)
     NodeDefaultTrustStore(cells/BIRKT40Cell02|security.xml#KeyStore_1134610924377)
     ```
   - List specific trust or key managers. Be sure to display the object name for the trust managers. You will need the object name for the SSL configuration because you can specify multiple trust manager instances.

     – Using Jacl:

     ```
     $AdminTask listTrustManagers {-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -displayObjectName true }
     ```

     – Using Jython:

     ```
     AdminTask.listTrustManagers ('[-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -displayObjectName true]')
     ```

     Example output:

     ```
     IbmX509(cells/BIRKT40Cell02|security.xml#TrustManager_1)
     IbmPKIX(cells/BIRKT40Cell02|security.xml#TrustManager_2)
     IbmX509(cells/BIRKT40Cell02|security.xml#TrustManager_1134610924357)
     IbmPKIX(cells/BIRKT40Cell02|security.xml#TrustManager_1134610924377)
     ```

2. Create the node-scoped SSL configuration in interactive mode. Now that we have the information we need to choose from, we need to decide if these objects are sufficient or if we need to create new ones. For now, we will reuse what we've already got in the configuration and save creating new instances to task documents specific to those objects.
   - Using Jacl:

     ```
     $AdminTask createSSLConfig -interactive
     ```

   - Using Jython:

     ```
     AdminTask.createSSLConfig ('[-interactive]')
     ```

   Example output:

   ```
   Create a SSL Configuration.

   *SSL Configuration Alias (alias): BIRKT40Node02SSLConfig
   Management Scope Name (scopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
   Client Key Alias (clientKeyAlias): default
   Server Key Alias (serverKeyAlias): default
   SSL Type (type): [JSSE]
   Client Authentication (clientAuthentication): [false]
   Security Level of the SSL Configuration (securityLevel): [HIGH]
   Enabled Ciphers SSL Configuration (enabledCiphers):
   JSSE Provider (jsseProvider): [IBMJSSE2]
   Client Authentication Support (clientAuthenticationSupported): [false]
   SSL Protocol (sslProtocol): [SSL_TLS]
   Trust Manager Object Names (trustManagerObjectNames): (cells/BIRKT40Cell02|security.xml#TrustManager_1)
   *Trust Store Name (trustStoreName): NodeDefaultTrustStore
   Trust Store Scope (trustStoreScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
   *Key Store Name (keyStoreName): NodeDefaultKeyStore
   Key Store Scope Name (keyStoreScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
   Key Manager Name (keyManagerName): IbmX509
   Key Manager Scope Name (keyManagerScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02

   Create SSL Configuration

   F (Finish)
   C (Cancel)

   Select [F, C]: [F] F
   WASX7278I: Generated command line: $AdminTask createSSLConfig {-alias BIRKT40Node02SSLConfig -scopeName
   (cell):BIRKT40Cell02:(node):BIRKT40Node02 -clientKeyAlias default -serverKeyAlias default
   -trustManagerObjectNames (cells/BIRKT40Cell02|security.xml#TrustManager_1) -trustStoreName
   NodeDefaultTrustStore -trustStoreScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -keyStoreName
   NodeDefaultKeyStore -keyStoreScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -keyManagerName
   IbmX509 -keyManagerScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 }
   ```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.
4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

The name of the SSL configuration object that you created, for example, (`cells/BIRKT40Cell02|security.xml#SSLConfig_1136652770753`), appears in the `security.xml` file.

Example `security.xml` file output:

```
<repertoire xmi:id="SSLConfig_1136652770753" alias="BIRKT40Node02SSLConfig" type="JSSE"
managementScope="ManagementScope_1134610924357">
<setting xmi:id="SecureSocketLayer_1136652770924" clientKeyAlias="default" serverKeyAlias="default"
clientAuthentication="false" securityLevel="HIGH"  jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS"
keyStore="KeyStore_1134610924357" trustStore="KeyStore_1134610924377" trustManager="TrustManager_1"
keyManager="KeyManager_1134610924357"/>
</repertoire>
```

Once you create the SSL configuration object, the next step is to use it. There are several different ways that you can associate SSL configurations with protocols, for example:

- Set the SSL configuration on the thread programmatically.
- Associate the SSL configuration with an outbound protocol or a target host and port.
- Directly associating the SSL configuration using the alias.
- Centrally managing the SSL configurations by associating them with SSL configuration groups or zones so that they are used based upon the group from where the end point exists.

Creating self-signed certificates using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

You can create self-signed certificates using the scripting and the AdminTask object. You can run the commands in interactive or batch mode. Interactive mode provides a way to discover the flags that you need to run the task in batch mode.

Certificates reside inside of key stores. To run the commands, you will need the name of the key store to be supplied. Use the **listKeyStore** command of the AdminTask object to get a list of key stores. If you need a new key store, use the **createKeyStore** command of the AdminTask object.

To create a personal key store, use the following examples:

- Interactive mode:
  - Using Jacl:

    ```
    $AdminTask createSelfSignedCertificate -interactive
    ```

  - Using Jython:

    ```
    AdminTask.createSelfSignedCertificate ('[-interactive]')
    ```

    Example output:

    ```
    *Key Store Name (keyStoreName): keyStore
    Key Store Scope Name (keyStoreScope):
    *Certificate Alias (certificateAlias): newCert
    "Certificate Version" (certificateVersion): 3
    *Key Size (certificateSize): [1024]
    *Common Name (certificateCommonName): localhost
    *Organization (certificateOrganization): workgroup
    Organizational Unit (certificateOrganizationalUnit): testing
    certLocality (certificateLocality): austin
    State (certificateState): Texas
    Zip (certificateZip): 78757
    Country (certificateCountry): [US]
    Validity Period (certificateValidDays): [365]
    Create Self-Signed Certificate

    F (Finish)
    C (Cancel)

    Select [F, C]: [F]

    WASX7278I: Generated command line: $AdminTask createSelfSignedCertificate
    {-keyStoreName keyStore -certificateAlias newCert -certificateVersion 3
    -certificateCommonName localhost -certificateOrganization ibm
    -certificateOrganizationalUnit testing -certificateLocality austin
    -certificateState Texas -certificateZip 78757 }
    true
    ```

    At the end of the output, the batch mode parameters are provided.

- Batch mode:
  - Using Jacl:

```
$AdminTask createSelfSignedCertificate {-keyStoreName keyStore
-certificateAlias newCert -certificateVersion 3 -certificateCommonName localhost
-certificateOrganization ibm -certificateOrganizationalUnit testing
-certificateLocality austin -certificateState Texas -certificateZip 78757 }
```

– Using Jython:

```
AdminTask.createSelfSignedCertificate ('[-keyStoreName keyStore
-certificateAlias newCert -certificateVersion 3 -certificateCommonName localhost
-certificateOrganization ibm -certificateOrganizationalUnit testing
-certificateLocality austin -certificateState Texas -certificateZip 78757]')
```

Automating SSL configurations using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

SSL configuration is needed for WebSphere to perform SSL connections with other servers. A SSL configuration can be configured through the Admin Console. But if an automated way to create a SSL configuration is desired then AdminTask should be used.

AdminTask can be used in a interactive mode and batch mode. For automation the batch mode options should be used. AdminTask batch mode can be called in a JACL or Python script. Interactive mode will step through all the parameter the task needs, requires ones are marked with a '*'. Before the interactive task executes the task it echoes the batch mode syntax of the task to the screen. This can be helpful when writing batch mode scripts.

There attributes needed to create an ssl configurations:

- A key store
- Default client certificate alias
- Default server certificate alias
- Trust store
- The handshake protocol
- The ciphers needed during handshake
- Supporting client authentication or not

If automating the creation of a SSL Configuration it may be needed to create some of the attribute values needed like the key store, trust store, key manager, and trust managers.

- To create a SSL configuration the createSSLConfig AdminTask can be used. To make changes to the SSL configurations use the modifySSLConfig AdminTask.
  – Interactive mode:

    Interactive mode steps you through all attributes and tell you the default value of the attribute if there is one. The default value is in '[]' on the prompt line. The actual flag used in batch mode is in '()' on each prompt line. If you are using the default value then the flag will not show up on the batch command line.

    Using Jacl:

    ```
    $AdminTask createSSLConfig -interactive
    ```

  – Using Jython:

    ```
    AdminTask.createSSLConfig ('[interactive]')
    ```

  Example output:

```
*SSL Configuration Alias (alias): testSSLConfig
Management Scope Name (scopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
Client Key Alias (clientKeyAlias): clientCert
Server Key Alias (serverKeyAlias): serverCert
SSL Type (type): [JSSE]
Client Authentication (clientAuthentication): [false]
Security Level of the SSL Configuration (securityLevel): [HIGH] HIGH
```

```
Enabled Ciphers SSL Configuration (enabledCiphers):
JSSE Provider (jsseProvider): [IBMJSSE2]
Client Authentication Support (clientAuthenticationSupported): [false]
SSL Protocol (sslProtocol): [SSL_TLS] SSL_TLS
Trust Manager Object Names (trustManagerObjectNames):
*Trust Store Name (trustStoreName): testTrustStore
Trust Store Scope (trustStoreScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
*Key Store Name (keyStoreName): testKeyStore
Key Store Scope Name (keyStoreScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
Key Manager Name (keyManagerName): IbmX509
Key Manager Scope Name (keyManagerScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01

Create SSL Configuration

F (Finish)
C (Cancel)

Select [F, C]: [F]
WASX7278I: Generated command line: $AdminTask createSSLConfig {-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01 }
(cells/HOSTNode01Cell|security.xml#SSLConfig_1137687301834)
```

At the end of the output, the batch mode parameters are provided.

– Batch mode:

  Using Jacl:

```
$AdminTask createSSLConfig {-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01}
```

– Using Jython:

```
AdminTask.createSSLConfig ('[-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01]')
```

Example output:

```
(cells/HOSTNode01Cell|security.xml#SSLConfig_1137687301834)
```

- Key Stores and Trust Stores The key store and trust store may already exist or a new one may need to be created. To create a new key store or trust store use the createKeyStore AdminTask. It will create a key store file and store the configuration object in the system configuration. A trust store is just a key store that usually only has signer certificates in it. To create a key store enter:

  – Using Jacl:

```
$AdminTask createKeyStore {-keyStoreName testKeyStore  -keyStoreType PKCS12
-keyStoreLocation $(USER_INSTALL_ROOT)\testKeyStore.p12 -keyStorePassword abcd
-keyStorePasswordVerify abcd  -keyStoreIsFileBased true -keyStoreReadOnly false}
```

  – Using Jython:

```
AdminTask.createKeyStore ('[-keyStoreName testKeyStore  -keyStoreType PKCS12
-keyStoreLocation $(USER_INSTALL_ROOT)\testKeyStore.p12 -keyStorePassword abcd
-keyStorePasswordVerify abcd  -keyStoreIsFileBased true -keyStoreReadOnly false]')
```

  To populate the key store with certificates see “Managing Certificates using AdminConsole and Admin Task” The key store and trust store are required to create a SSL configuration. Use the ‘-keyStoreName’ and ‘-trustStoreName’ flags on the createSSLConfig. There scopes can be added with the ‘-keyStoreScope’ flag and ‘-trustStoreScope’ flags.

- Key Manager Key manager are used to determine how a certificate is selected. The IbmX509 key manager is in the security configuration by default. If a different key manager is needed then use createKeyManager AdminTask to create it. To create a key manager enter:
  - Using Jacl:

    ```
    $AdminTask createKeyManager {-name testKeyManager
    -scopeName  (cell):HOSTNode01Cell:(node):HOSTNode01
    -provider IBMJSSE2 -algorithm specialAlgorithm }
    ```

  - Using Jython:

    ```
    AdminTask.createKeyManager ('[-name testKeyManager
    -scopeName  (cell):HOSTNode01Cell:(node):HOSTNode01
    -provider IBMJSSE2 -algorithm specialAlgorithm]')
    ```

  To supply a key manager on the createSSLConfig AdminTask use the '-keyManagerName' along with the '-keyManagerScope' flag.
- Trust Manager Trust managers are use to determine how trust is established during ssl communication. The IbmX509 and IbmPKIX are trust managers are in the security configuration by default. If a different or additional trust manager is needed then use the createTrustManger AdminTask to create it. To create a trust manager enter:
  - Using Jacl:

    ```
    $AdminTask createTrustManager {-name testTrustManager
    -scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
    -provider IBMJSSE2 -algorithm specialAlgorithm }
    ```

  - Using Jython:

    ```
    AdminTask.createTrustManager ('[-name testTrustManager
    -scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
    -provider IBMJSSE2 -algorithm specialAlgorithm]')
    ```

  The SSL Configuration can have multiple trust managers. To supply multiple trust managers give a comma separated list of the trust managers configuration IDs with the `-trustManagerObjectNames` flag. When you create a trust manager the configuration object ID is returned. To get a list of trust managers object IDs use the **listTrustManagers** command of the AdminTask object with the `-displayObjectName true` flag. For example:

```
wsadmin>$AdminTask listTrustManagers -interactive
List Trust Managers

List trust managers.

Management Scope Name (scopeName):
Display list in ObjectName Format (displayObjectName): [false] true

List Trust Managers

F (Finish)
C (Cancel)

Select [F, C]: [F]
Inside generate script command
WASX7278I: Generated command line: $AdminTask listTrustManagers {-displayObjectName true }
IbmX509(cells/IBM-0AF8DABCF16Node01Cell|security.xml#TrustManager_IBM-0AF8DABCF16Node01_1)
IbmPKIX(cells/IBM-0AF8DABCF16Node01Cell|security.xml#TrustManager_IBM-0AF8DABCF16Node01_2)
```

Updating default key store passwords using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

When you install WebSphere Application Server, each server creates a key store and trust store for the default SSL configuration with the default password `WebAS`. To protect the security of the key store files and the SSL configuration, you must change the password. The following examples update the default password:

- Change multiple key stores passwords. The **changeMultipleKeyStorePasswords** command updates all of the key stores that have the same password. For example:
  - Using Jacl:

    ```
    $AdminTask changeMultipleKeyStorePasswords  {-keyStorePassword WebAS
    -newKeyStorePassword  secretPwd   -newKeyStorePasswordVerify secretPwd}
    ```

  - Using Jython:

    ```
    AdminTask.changeMultipleKeyStorePasswords  ['(-keyStorePassword WebAS
    -newKeyStorePassword secretPwd -newKeyStorePasswordVerify secretPwd]')
    ```

- Change the password of a single key store. The **changeKeyStorePassword** command updates the password of an individual key store. For example:
  - Using Jacl:

    ```
    $AdminTask changeKeyStorePassword {-keyStoreName testKS
    -keyStoreScope (cell):localhost:(server):server1
    -keyStorePassword WebAS -newKeyStorePassword secretPwd
    -newKeyStorePasswordVerify secretPwd}
    ```

  - Using Jython:

    ```
    AdminTask.changeKeyStorePassword ('[-keyStoreName testKS
    -keyStoreScope (cell):localhost:(server):server1
    -keyStorePassword WebAS -newKeyStorePassword secretPwd
    -newKeyStorePasswordVerify secretPwd]')
    ```

Commands for the IdMgrConfig group of the AdminTask object

Use the commands in the IdMgrConfig group to configure the virtual member manager. The commands for this group do not require a target object. To see the additional commands related to the virtual member manager, see the Commands for the IdMgrRepositoryConfig group of the AdminTask object and the Commands for the IdMgrRealmConfig group of the AdminTask object articles.

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the IdMgrConfig group of the AdminTask object:

Table 24.

Command name:	Description:	Parameters and return values:	Examples:
createIdMgr Supported EntityType	The **createIdMgr Supported EntityType** command creates a supported entity type configuration. To validate the result, check for duplicate entity type names.	• Parameters: **- name** The name of the supported entity type. (String, required) **- defaultParent** The default parent node for the supported entity type. (String, required) **- rdnProperties** The RDN attribute name for the supported entity type in the entity domain name. Separate the RDN properties with semicolons (;). (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask createIdMgrSupported EntityType {-name entity1 -defaultParent node1}` • Using Jython string: `AdminTask.createIdMgrSupported EntityType ('[-name entity1 -defaultParent node1]')` • Using Jython list: `AdminTask.createIdMgrSupported EntityType (['-name', 'entity1', '-defaultParent', 'node1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createIdMgrSupported EntityType {-interactive}` • Using Jython string: `AdminTask.createIdMgrSupported EntityType ('[-interactive]')` • Using Jython list: `AdminTask.createIdMgrSupported EntityType (['-interactive'])`

Table 24. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteIdMgr Supported EntityType	The **deleteIdMgr Supported EntityType** command deletes the supported entity type configuration that you specify.	• Parameters: **- name** The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrSupported EntityType {-name entity1}` • Using Jython string: `AdminTask.deleteIdMgrSupported EntityType ('[-name entity1]')` • Using Jython list: `AdminTask.deleteIdMgr SupportedEntityType (['-name', 'entity1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgr SupportedEntityType {-interactive}` • Using Jython string: `AdminTask.deleteIdMgr SupportedEntityType ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgr SupportedEntityType (['-interactive'])`
getIdMgr Supported EntityType	The **getIdMgr Supported EntityType** command returns the configuration of the supported entity type that you specify.	• Parameters: **- name** The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required) • Returns: A hash map that has the parameters of the **createIdMgrSupportedEntityType** command as keys. Since the rdnProperties parameter has multiple values, its values are returned in a list.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrSupported EntityType {-name entity1}` • Using Jython string: `AdminTask.getIdMgrSupported EntityType ('[-name entity1]')` • Using Jython list: `AdminTask.getIdMgrSupported EntityType (['-name', 'entity1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrSupported EntityType {-interactive}` • Using Jython string: `AdminTask.getIdMgrSupported EntityType ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrSupported EntityType (['-interactive'])`

Table 24. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr Supported EntityTypes	The **listIdMgr Supported EntityTypes** command lists all of the supported entity types that are configured.	• Parameters: None • Returns: A list that contains the names of the supported entity types.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgr SupportedEntityTypes` • Using Jython string: `AdminTask.listIdMgr SupportedEntityTypes()` • Using Jython list: `AdminTask.listIdMgr SupportedEntityTypes()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrSupported EntityTypes {-interactive}` • Using Jython string: `AdminTask.listIdMgrSupported EntityTypes ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrSupported EntityTypes (['-interactive'])`
resetId MgrConfig	The **resetId MgrConfig** command resets the current configuration to the last configuration that was saved.	• Parameters: None • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask resetIdMgrConfig` • Using Jython string: `AdminTask.resetIdMgrConfig()` • Using Jython list: `AdminTask.resetIdMgrConfig()` **Interactive mode example usage:** • Using Jacl: `$AdminTask resetIdMgrConfig {-interactive}` • Using Jython string: `AdminTask.resetIdMgrConfig ('[-interactive]')` • Using Jython list: `AdminTask.resetIdMgrConfig (['-interactive'])`

Table 24. (continued)

Command name:	Description:	Parameters and return values:	Examples:
showId MgrConfig	The **showId MgrConfig** command returns the current configuration XML in string format.	• Parameters: None • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask showIdMgrConfig` • Using Jython string: `AdminTask.showIdMgrConfig()` • Using Jython list: `AdminTask.showIdMgrConfig()` **Interactive mode example usage:** • Using Jacl: `$AdminTask showIdMgrConfig {-interactive}` • Using Jython string: `AdminTask.showIdMgrConfig ('[-interactive]')` • Using Jython list: `AdminTask.showIdMgrConfig (['-interactive'])`
updateIdMgr Supported EntityType	The **updateIdMgr Supported EntityType** command updates the configuration that you specify for a supported entity type.	• Parameters: **- name** The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required) **- defaultParent** The default parent node for the supported entity type. (String, optional) **- rdnProperties** The RDN attribute name for the supported entity type in the entity domain name. To reset all the values of the rdnProperties parameter, specify a blank string (""). (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrSupported EntityType {-name entity1}` • Using Jython string: `AdminTask.updateIdMgrSupported EntityType ('[-name entity1]')` • Using Jython list: `AdminTask.updateIdMgrSupported EntityType (['-name', 'entity1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrSupported EntityType {-interactive}` • Using Jython string: `AdminTask.updateIdMgrSupported EntityType ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrSupported EntityType (['-interactive'])`

Commands for the IdMgrRepositoryConfig group of the AdminTask object

Use the commands in the IdMgrRepositoryConfig to configure the virtual member manager. The commands for this group do not require a target object. To see the additional commands related to the virtual member manager, see the “Commands for the IdMgrConfig group of the AdminTask object” on page 1528 and the “Commands for the IdMgrRealmConfig group of the AdminTask object” on page 1622 articles.

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the IdMgrRepositoryConfig group of the AdminTask object:

Table 25.

Command name:	Description:	Parameters and return values:	Examples:
addIdMgrLDAP BackupServer	The **addIdMgrLDAP BackupServer** command adds or updates backup LDAP servers.	• Parameters: **- id** The ID of the repository. (String, required) **- primary_host** The host name for the primary LDAP server. (String, required) **- host** The host name for the LDAP server. (String, required) **- port** The port number for the LDAP server. (Integer, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPBackup Server {-id id1 -primary_ host myprimaryhost -host myhost.ibm.com}` • Using Jython string: `AdminTask.addIdMgrLDAPBackup Server ('[-id id1 -primary_host myprimaryhost -host myhost.ibm.com]')` • Using Jython list: `AdminTask.addIdMgrLDAPBackup Server (['-id', 'id1', '-primary_host', 'myprimary host', '-host', 'myhost. ibm.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAP BackupServer {-interactive}` • Using Jython string: `AdminTask.addIdMgrLDAPBack upServer ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrLDAPBack upServer (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
addIdMgr LDAPEntity Type	The **addIdMgr LDAPEntity Type** command adds an LDAP entity type definition.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the entity type. (String, required) **- searchFilter** The search filter that you want to use to search the entity type. (String, optional) **- objectClasses** One or more object classes for the entity type. (String, required) **- objectClassesForCreate** The object class to use when an entity type is created. If the value of this parameter is the same as the objectClass parameter, you do not need to specify this parameter. (String, optional) **- searchBases** The search base or bases to use while searching the entity type. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPEntity Type {-id id1 -name name1 -objectClasses objectclass}` • Using Jython string: `AdminTask.addIdMgrLDAPEntity Type ('[-id id1 -name name1 -objectClasses objectclass]')` • Using Jython list: `AdminTask.addIdMgrLDAPEntity Type (['-id', 'id1', '-name', 'name1', '-objectClasses', 'objectclass'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAP EntityType {-interactive}` • Using Jython string: `AdminTask.addIdMgrLDAP EntityType ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrLDAP EntityType (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
addId MgrLDAP EntityType RDNAttr	The **addId MgrLDAP EntityType RDNAttr** command adds RDN attribute configuration to an LDAP entity type definition.	• Parameters: **- id** The ID of the repository. (String, required) **- entityTypeName** The name of the entity type. (String, required) **- name** The attribute name that is used to build the relative distinguished name (RDN) for the entity type. (String, required) **- objectClass** The object class to use for the entity type for the relative distinguished name (RDN) attribute name that you specify. Use this parameter to map one entity type to multiple structural object classes. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPEntity TypeRDNAttr {-id id1 -entityTypeName entitytype -name name1}` • Using Jython string: `AdminTask.addIdMgrLDAPEntity TypeRDNAttr ('[-id id1 -entityTypeName entitytype -name name1]')` • Using Jython list: `AdminTask.addIdMgrLDAPEntity TypeRDNAttr (['-id', 'id1', '-entityTypeName', 'entity type', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPEntity TypeRDNAttr {-interactive}` • Using Jython string: `AdminTask.addIdMgrLDAPEntity TypeRDNAttr ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrLDAPEntity TypeRDNAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
addIdMgr LDAPGroup Dynamic Member Attr	The **addIdMgr LDAPGroup Dynamic Member Attr** command adds a dynamic member attribute configuration to an LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the LDAP attribute that is used as the group member attribute. For example, `member` or `uniqueMember`. (String, required) **- objectClass** The group object class that contains the member attribute. For example, `groupOfNames` or `groupOfUnqiueNames`. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPGroup DynamicMemberAttr {-id id1 -name name1 -objectClass objectclass}` • Using Jython string: `AdminTask.addIdMgrLDAPGroup DynamicMemberAttr ('[-id id1 -name name1 -objectClass objectclass]')` • Using Jython list: `AdminTask.addIdMgrLDAPGroup DynamicMemberAttr (['-id', 'id1', '-name', 'name1', '-objectClass', 'objectclass'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPGroup DynamicMemberAttr {-interactive}` • Using Jython string: `AdminTask.addIdMgrLDAPGroup DynamicMemberAttr ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- scope The scope of the member attribute. The valid values for this parameter include the following: • direct - The member attribute only contains direct members, therefore, this value refers to the member directly contained by the group and not contained through the nested group. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct member of Group1 but User1 is not a direct member of Group1. Both `member` and `uniqueMember` are direct member attributes. • nested - The member attribute that contains the direct members and the nested members.	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		• all - The member attribute that contains the direct members, the nested members, and the dynamic members. For example, the ibm-allMembers attribute which is supported by the IBM Tivoli Directory Server. (String, optional) **- dummyMember** Indicates that if you create a group without specifying a member, a dummy member will be filled in to avoid creating an exception about missing a mandatory attribute. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
addIdMgr LDAPGroup MemberAttr	The **addIdMgr LDAPGroup MemberAttr** command adds a member attribute configuration to an LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the LDAP attribute that is used as the group member attribute. For example, `member` or `uniqueMember`. (String, required) **- objectClass** The group object class that contains the member attribute. For example, `groupOfNames` or `groupOfUnqiueNames`. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPGroup MemberAttr {-id id1 -name name1}` • Using Jython string: `AdminTask.addIdMgrLDAPGroup MemberAttr ('[-id id1 -name name1]')` • Using Jython list: `AdminTask.addIdMgrLDAPGroup MemberAttr (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPGroup MemberAttr {-interactive}` • Using Jython string: `AdminTask.addIdMgrLDAPGroup MemberAttr ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrLDAPGroup MemberAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- scope The scope of the member attribute. The valid values for this parameter include the following: • direct - The member attribute only contains direct members, therefore, this value refers to the member directly contained by the group and not contained through the nested group. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct member of Group1 but User1 is not a direct member of Group1. Both `member` and `uniqueMember` are direct member attributes. • nested - The member attribute that contains the direct members and the nested members.	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		• all - The member attribute that contains the direct members, the nested members, and the dynamic members. For example, the ibm-allMembers attribute which is supported by the IBM Tivoli Directory Server. (String, optional) **- dummyMember** Indicates that if you create a group without specifying a member, a dummy member will be filled in to avoid creating an exception about missing a mandatory attribute. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
addId MgrLDAP Server	The **addId MgrLDAP Server** command adds an LDAP server to the LDAP repository ID that you specify.	• Parameters: **- id** The ID of the repository. (String, required) **- host** The host name for the primary LDAP server. (String, required) **- port** The port number for the LDAP server. (Integer, optional) **- bindDN** The binding distinguished name for the LDAP server. (String, optional) **- bindPassword** The binding password. (String, optional) **- authentication** Indicates the authentication method to use. The default value is `simple`. Valid values include: `none` or `strong`. (String, optional) **- referal** The LDAP referral. The default value is `ignore`. Valid values include: `follow`, `throw`, or `false`. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAPServer {-id id1 -host myhost.ibm.com}` • Using Jython string: `AdminTask.addIdMgrLDAPServer ('[-id id1 -host myhost.ibm.com]')` • Using Jython list: `AdminTask.addIdMgrLDAPServer (['-id', 'id1', '-host', 'myhost.ibm.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrLDAP Server {-interactive}` • Using Jython string: `AdminTask.addIdMgrLDAP Server ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrLDAP Server (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- derefAliases Controls how aliases are dereferenced. The default value is `always`. Valid values include: • `never` - never deference aliases • `finding` - deferences aliases only during name resolution • `searching` - deferences aliases only after name resolution (String, optional) **- sslEnabled** Indicates to enable SSL or not. The default value is `false`. (Boolean, optional) **- connectionPool** The connection pool. The default value is `false`. (Boolean, optional) **- connectTimeout** The connection timeout in seconds. The default value is `0`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- ldapServerType The type of LDAP server being used. The default value is `IDS51`. (String, optional) **- sslConfiguration** The SSL configuration. (String, optional) **- certificateMapMode** Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is `exactdn`. To use the certificate filter for the mapping, specify `certificatefilter`. (String, optional) **- certificateFilter** If `certificateMapMode` has the value `certificatefilter`, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
addIdMgr Repository BaseEntry	The **addIdMgr Repository BaseEntry**command adds a base entry to the specified repository.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The distinguished name of a base entry. (String, required) **- nameInRepository** The distinguished name in the repository that uniquely identifies the base entry name. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrRepository BaseEntry {-id id1 -name name1}` • Using Jython string: `AdminTask.addIdMgrRepository BaseEntry ('[-id id1 -name name1]')` • Using Jython list: `AdminTask.addIdMgrRepository BaseEntry (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrRepository BaseEntry {-interactive}` • Using Jython string: `AdminTask.addIdMgrRepository BaseEntry ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrRepository BaseEntry (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createId MgrDB Repository	The **createId MgrDB Repository** command creates a database repository configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- dataSourceName** The name of the data source. The default value is `jdbc/wimDS`. (String, required) **- databaseType** The type of the database. The default value is `DB2`. (String, required) **- dbURL** The URL of the database. (String, required) **- dbAdminId** The database administrator ID. (String, required if database type is not Cloudscape.) **- dbAdminPassword** The database administrator password. (String, required if database type is not Cloudscape.) **- adapterClassName** The default value is `com.ibm.ws.wim.adapter.db.DBAdapter`. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createIdMgrDB Repository {-id id1 -dataSourceName datasource name -databaseType DB2}` • Using Jython string: `AdminTask.createIdMgrDB Repository ('[-id id1 -dataSourceName data sourcename -databaseType DB2]')` • Using Jython list: `AdminTask.createIdMgrDB Repository (['-id', 'id1', '-dataSourceName', 'data sourcename', '-databaseType', 'DB2'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createIdMgrDB Repository {-interactive}` • Using Jython string: `AdminTask.createIdMgrDB Repository ('[-interactive]')` • Using Jython list: `AdminTask.createIdMgrDB Repository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- JDBCDriverClass The JDBC driver class name. (String, optional) **- supportSorting** Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional) **- supportTransaction** Indicates if transactions are supported or not. The default value is `false`. (Boolean, optional) **- isExtIdUnique** Specifies if the external ID is unique. The default value is `true`. (Boolean, optional) **- supportExternalName** Indicates if external names are supported or not. The default value is `false`. (Boolean, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- entityRetrievalLimit Indicates the value of the retrieval limit on database entries. The default value is `200`. (Integer, optional) **- saltLength** The salt length in bits. The default value is `12`. (Integer, optional) **- encryptionKey** The default value is `rZ15ws0ely9yHk3zCs3sTMv/ho8fY17s`. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createId MgrFile Repository	The **createId MgrFile Repository** command creates a file repository configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- messageDigest Algorithm** The message digest algorithm that will be used for hashing the password. The default value is `SHA-1`. Valid values include the following: `SHA-245`, `SHA-384`, or `SHA-512`.(String, required) **- adapterClassName** The default value is `com.ibm.ws.wim.adapter.file.was.FileAdapter`. (String, optional) **- supportPaging** Indicates if paging is supported or not. The default value is `false`. (Boolean, optional) **- supportSorting** Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createIdMgrFile Repository {-id id1 -messageDigestAlgorithm SHA-245}` • Using Jython string: `AdminTask.createIdMgrFile Repository ('[-id id1 -messageDigestAlgorithm SHA-245]')` • Using Jython list: `AdminTask.createIdMgrFile Repository (['-id', 'id1', '-messageDigestAlgorithm', 'SHA-245'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createIdMgrFile Repository {-interactive}` • Using Jython string: `AdminTask.createIdMgrFile Repository ('[-interactive]')` • Using Jython list: `AdminTask.createIdMgrFile Repository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- supportTransaction Indicates if transaction is supported or not. The default value is `false`. (Boolean, optional) **- isExtIdUnique** Specifies if the external ID is unique or not. The default value is `true`. (Boolean, optional) **- supportExternalName** Indicates if external names are supported or not. The default value is `false`. (Boolean, optional) **- baseDirectory** The base directory where the fill will be created in order to store the data. The default is to be dynamically built during run time using user.install.root and cell name. (String, optional) **- fileName** The file name of the repository. The default value is `fileRegistry.xml`. (String, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- saltLength The salt length of the randomly generated salt for password hashing. The default value is 12. (Integer, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createId MgrLDAP Repository	The **create IdMgrLDAP Repository** command creates an LDAP repository configuration.	• Parameters: **- id** The unique identifier for the repository. (String, required) **- ldapServerType** The type of LDAP server that is being used. The default value is `IDS51`. (String, required) **- adapterClassName** The default value is `com.ibm.ws.wim.adapter.db.DBAda pter`. (String, optional) **- supportSorting** Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional) **- supportPaging** Indicates if paging is supported or not. The default value is `false`. (Boolean, optional) **- supportTransaction** Indicates if transactions are supported or not. The default value is `false`. (Boolean, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createIdMgrLDAP Repository {-id id1 -ldapServerType IDS51}` • Using Jython string: `AdminTask.createIdMgrLDAP Repository ('[-id id1 -ldapServerType IDS51]')` • Using Jython list: `AdminTask.createIdMgrLDAP Repository (['-id', 'id1', '-ldapServerType', 'IDS51'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createIdMgrLDAP Repository {-interactive}` • Using Jython string: `AdminTask.createIdMgrLDAP Repository ('[-interactive]')` • Using Jython list: `AdminTask.createIdMgrLDAP Repository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- isExtIdUnique Specifies if the external ID is unique. The default value is `true`. (Boolean, optional) **- supportExternalName** Indicates if external names are supported or not. The default value is `false`. (Boolean, optional) **- authentication** Indicates the authentication method to use. The default value is `simple`. Valid values include: `none` or `strong`. (String, optional) **- referal** The LDAP referral. The default value is `ignore`. Valid values include: `follow`, `throw`, or `false`. (String, optional) **- sslEnabled** Indicates to enable SSL or not. The default value is `false`. (Boolean, optional) **- sslConfiguration** The SSL configuration. (String, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- connectionPool The connection pool. The default value is `false`. (Boolean, optional) **- translateRDN** Indicates to translate RDN or not. The default value is `false`. (Boolean, optional) **- searchTimeLimit** The value of search time limit. (Integer, optional) **- searchCountLimit** The value of search count limit. (Integer, optional) **- searchPageSize** The value of search page size. (Integer, optional) **- returnToPrimaryServer** (Integer, optional) **- primaryServerQueryTimeInterval** (Integer, optional) **- default** If you set this parameter to `true`, the default values will be set for the remaining configuration properties of the LDAP repository. (Boolean, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteId MgrLDAP EntityType	The **deleteId MgrLDAP EntityType** command deletes the LDAP entity type configuration data for a specified entity type for a specific LDAP repository.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the entity type. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP EntityType {-id id1 -name name1}` • Using Jython string: `AdminTask.deleteIdMgrLDAP EntityType ('[-id id1 -name name1]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP EntityType (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP EntityType {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrLDAP EntityType ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP EntityType (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteId MgrLDAP EntityType RDNAttr	The **deleteId MgrLDAP EntityType RDNAttr** command deletes the relative distinguished name (RDN) attribute configuration from an LDAP entity type configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- entityTypeName** The name of the entity type. (String, required) **- name** The attribute name that is used to build the relative distinguished name (RDN) for the entity type. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAPEntity TypeRDNAttr {-id id1 -name name1 -entityTypeName entityType}` • Using Jython string: `AdminTask.deleteIdMgrLDAP EntityTypeRDNAttr ('[-id id1 -name name1 -entityType Name entityType]')` • Using Jython list: `AdminTask.deleteIdMgrLDAPEntity TypeRDNAttr (['-id', 'id1', '-name', 'name1', '-entity TypeName', 'entityType'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAPEntity TypeRDNAttr {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrLDAPEntity TypeRDNAttr ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrLDAPEntity TypeRDNAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteId MgrLDAP GroupConfig	The **deleteId MgrLDAP GroupConfig** command deletes the LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP GroupConfig {-id id1}` • Using Jython string: `AdminTask.deleteIdMgrLDAP GroupConfig ('[-id id1]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP GroupConfig (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP GroupConfig {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrLDAP GroupConfig ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP GroupConfig (['-interactive'])`
deleteIdMgr LDAPGroup MemberAttr	The **deleteIdMgr LDAPGroup MemberAttr** command deletes a member attribute configuration from an LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP GroupMemberAttr {-id id1}` • Using Jython string: `AdminTask.deleteIdMgrLDAP GroupMemberAttr ('[-id id1]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP GroupMemberAttr (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP GroupMemberAttr {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrLDAP GroupMemberAttr ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP GroupMemberAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteIdMgr LDAPGroup Dynamic MemberAttr	The **deleteIdMgr LDAPGroup Dynamic MemberAttr** command deletes a dynamic member attribute configuration from an LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the LDAP attribute that is used as the group member attribute. For example, `memberURL`. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP GroupDynamicMemberAttr {-id id1 -name name1}` • Using Jython string: `AdminTask.deleteIdMgrLDAP GroupDynamicMemberAttr ('[ -id id1 -name name1]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP GroupDynamicMemberAttr (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAPGroup DynamicMemberAttr {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrLDAPGroup DynamicMemberAttr ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteId MgrLDAP Server	The **deleteId MgrLDAP Server** command deletes the configuration for the LDAP server that you specify from the LDAP repository ID that you specify.	• Parameters: **- id** The ID of the repository. (String, required) **- host** The host name for the primary LDAP server. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP Server {-id id1 -host myhost.ibm.com}` • Using Jython string: `AdminTask.deleteIdMgrLDAP Server ('[-id id1 -host myhost.ibm.com]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP Server (['-id', 'id1', '-host', 'myhost.ibm.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrLDAP Server {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrLDAP Server ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrLDAP Server (['-interactive'])`
deleteIdMgr Repository	The **deleteIdMgr Repository** command deletes a repository that you specify.	• Parameters: **- id** The ID of the repository. Valid values include existing repository IDs. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgr Repository {-id id1}` • Using Jython string: `AdminTask.deleteIdMgr Repository ('[-id id1]')` • Using Jython list: `AdminTask.deleteIdMgr Repository (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrRepos itory {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrReposi tory ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrReposi tory (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteIdMgr Repository BaseEntry	The **deleteIdMgr Repository BaseEntry** command deletes a base entry from the specified repository.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The distinguished name of a base entry. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrRepository BaseEntry {-id id1 -name name1}` • Using Jython string: `AdminTask.deleteIdMgrRepository BaseEntry ('[-id id1 -name name1]')` • Using Jython list: `AdminTask.deleteIdMgrRepository BaseEntry (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrRepository BaseEntry {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrRepository BaseEntry ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrRepository BaseEntry (['-interactive'])`
getIdMgr LDAPAttr Cache	The **getIdMgr LDAPAttr Cache** command returns the LDAP attribute cache configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map with the parameters of the **setIdMgr LDAPAttr Cache** command as keys.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPAttr Cache {-id id1}` • Using Jython string: `AdminTask.getIdMgrLDAPAttr Cache ('[-id id1]')` • Using Jython list: `AdminTask.getIdMgrLDAPAttr Cache (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAP AttrCache {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPAttr Cache ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPAttr Cache (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr LDAP ContextPool	The **getIdMgr LDAP Context Pool** command returns the LDAP context pool configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map that includes the parameters of the **setIdMgrLDAPContextPool** command as the keys.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPContext Pool {-id id1}` • Using Jython string: `AdminTask.getIdMgrLDAPContext Pool ('[-id id1]')` • Using Jython list: `AdminTask.getIdMgrLDAPContext Pool (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPCon textPool {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPCon textPool ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPCon textPool (['-interactive'])`
getIdMgr LDAP EntityType	The **getIdMgr LDAP EntityType** command returns the LDAP entity type configuration data.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the entity type. (String, required) • Returns: A hash map with the keys the same as the property name of the **addIdMgr LDAP EntityType** command. Multi-valued parameters are returned as list.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPEntity Type {-id id1 -name name1}` • Using Jython string: `AdminTask.getIdMgrLDAPEntity Type ('[-id id1 -name name1]')` • Using Jython list: `AdminTask.getIdMgrLDAPEntity Type (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPEn tityType {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPEn tityType ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPEn tityType (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr LDAPEntity TypeRDNAttr	The **getIdMgr LDAPEntity TypeRDNAttr** command returns the relative distinguished name (RDN) attribute configuration for an LDAP entity type definition.	• Parameters: **- id** The ID of the repository. (String, required) **- entityTypeName** The name of the entity name. (String, required) • Returns: A hash map with the RDN attribute names as the key. If the object class is set, the value of the key will be set to the value of the object class. Otherwise, the value will be null.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPEntity TypeRDNAttr {-id id1 -entityTypeName name1}` • Using Jython string: `AdminTask.getIdMgrLDAPEntity TypeRDNAttr ('[-id id1 -entityTypeName name1]')` • Using Jython list: `AdminTask.getIdMgrLDAPEntity TypeRDNAttr (['-id', 'id1', '-entityTypeName', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPEntity TypeRDNAttr {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPEntity TypeRDNAttr ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPEntity TypeRDNAttr (['-interactive'])`
getIdMgr LDAPGroupConfig	The **getIdMgr LDAPGroupConfig** command returns the LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map that contains the parameters of the **setIdMgr LDAPGroup Config** command as the keys.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPGroup Config {-id id1}` • Using Jython string: `AdminTask.getIdMgrLDAPGroup Config ('[-id id1]')` • Using Jython list: `AdminTask.getIdMgrLDAPGroup Config (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPGroup Config {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPGroup Config ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPGroup Config (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr LDAPGroup Dynamic Member Attrs	The **getIdMgr LDAPGroup Dynamic Member Attrs** command returns the dynamic member attribute configuration from the LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map list that contains the parameters of the **addIdMgr LDAPGroup Dynamic MemberAttr** command as the keys.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPGroupDynamic MemberAttrs {-id id1}` • Using Jython string: `AdminTask.getIdMgrLDAPGroupDynamic MemberAttrs ('[-id id1]')` • Using Jython list: `AdminTask.getIdMgrLDAPGroupDynamic MemberAttrs (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPGroup DynamicMemberAttrs {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPGroup DynamicMemberAttrs ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPGroup DynamicMemberAttrs (['-interactive'])`
getIdMgr LDAPGroup MemberAttrs	The **getIdMgr LDAPGroup MemberAttrs** command returns the member attribute configuration for the LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map list that contains the parameters of the **addIdMgr LDAPGroup MemberAttr** command as the keys.	**Batch mode example usage:** **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPGroup MemberAttrs {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPGroup MemberAttrs ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPGroup MemberAttrs (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr LDAPSearch ResultCache	The **getIdMgr LDAPSearch ResultCache** command returns the LDAP search result cache configuration.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map with the parameters of the **setIdMgr LDAPSearch ResultCache** command as the keys.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAP SearchResultCache {-id id1}` • Using Jython string: `AdminTask.getIdMgrLDAP SearchResultCache ('[-id id1]')` • Using Jython list: `AdminTask.getIdMgrLDAPSear chResultCache (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPSea rchResultCache {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPSear chResultCache ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPSe archResultCache (['-interactive'])`
getIdMgr LDAPServer	The **getIdMgr LDAPServer** command returns the configuration for the LDAP server that you specify for the LDAP repository ID that you specify.	• Parameters: **- id** The ID of the repository. (String, required) **- host** The host name for the primary LDAP server. (String, required) • Returns: A hash map with the keys the same as the parameter names for the **addIdMgr LDAPServer** command.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPServer {-id id1 -host myhost.ibm.com}` • Using Jython string: `AdminTask.getIdMgrLDAPServer ('[-id id1 -host myhost.ibm.com]')` • Using Jython list: `AdminTask.getIdMgrLDAPServer (['-id', 'id1', '-host', 'myhost.ibm.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrLDAPServer {-interactive}` • Using Jython string: `AdminTask.getIdMgrLDAPServer ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrLDAPServer (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr Repository	The **getIdMgr Repository** command returns the configuration of the specified repository.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map. The keys will vary depending on the type of repository.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrRepos itory {-id id1}` • Using Jython string: `AdminTask.getIdMgrReposi tory ('[-id id1]')` • Using Jython list: `AdminTask.getIdMgrReposi tory (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrRepos itory {-interactive}` • Using Jython string: `AdminTask.getIdMgrReposi tory ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrReposi tory (['-interactive'])`
listIdMgr Custom Properties	The **listIdMgr Custom Properties**command returns a list of custom properties for the repository that you specify.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map that contains keys as the custom property names and values as custom property values.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrCustomPro perties {-id id1}` • Using Jython string: `AdminTask.listIdMgrCustom Properties ('[-id id1]')` • Using Jython list: `AdminTask.listIdMgrCustom Properties (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrCustom Properties {-interactive}` • Using Jython string: `AdminTask.listIdMgrCustom Properties ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrCustom Properties (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr LDAPBackupServers	The **listIdMgr LDAPBackupServers**command returns a list of the backup LDAP server or servers.	• Parameters: **- id** The ID of the repository. (String, required) **- primary_host** The host name for the primary LDAP server. (String, required) • Returns: A list of hash maps. The hash maps contains the names of the backup servers as the key and the port numbers as the value. Returning all the data in a hash map does not maintain the order of backup servers.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrLDAPBackupServer {-id id1 -primary_host hostname}` • Using Jython string: `AdminTask.listIdMgrLDAPBackupServer ('[-id id1 -primary_host hostname]')` • Using Jython list: `AdminTask.listIdMgrLDAPBackupServer (['-id', 'id1', '-primary_host', 'hostname'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrLDAPBackupServer {-interactive}` • Using Jython string: `AdminTask.listIdMgrLDAPBackupServer ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrLDAPBackupServer (['-interactive'])`
listIdMgr LDAPEntityTypes	The **listIdMgr LDAPEntityTypes** command lists the name of all of the configured LDAP entity type definitions.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A list that contains the names of the configured LDAP entity types.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrLDAPEntityType {-id id1}` • Using Jython string: `AdminTask.listIdMgrLDAPEntityType ('[-id id1]')` • Using Jython list: `AdminTask.listIdMgrLDAPEntityType (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrLDAPEntityType {-interactive}` • Using Jython string: `AdminTask.listIdMgrLDAPEntityType ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrLDAPEntityType (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr LDAP Servers	The **listIdMgr LDAP Servers** command lists all of the configured primary LDAP servers.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A list that contains the primary LDAP server names.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrLDAP Servers {-id id1}` • Using Jython string: `AdminTask.listIdMgrLDAP Servers ('[-id id1]')` • Using Jython list: `AdminTask.listIdMgrLDAP Servers (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrLDAP Servers {-interactive}` • Using Jython string: `AdminTask.listIdMgrLDAP Servers ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrLDAP Servers (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr Repositories	The **listIdMgr Repositories** command lists names and types of all configured repositories.	• Parameters: None • Returns: A hash map with key as the name of the repository and value as another hash map that includes the following keys: – repositoryType - The type of repository. For example, File, LDAP, DB, and so on. – specificRepositoryType - The specific type of repository. For example, LDAP, IDS51, NDS, and so on. – host - The host name where the repository resides. For File, it is LocalHost and for DB it is dataSourceName. This command will not return the Property Extension and Entry Mapping repository data.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrRepositories` • Using Jython string: `AdminTask.listIdMgrRepositories()` • Using Jython list: `AdminTask.listIdMgrRepositories()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrReposi tories {-interactive}` • Using Jython string: `AdminTask.listIdMgrReposit ories ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrReposi tories (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr Repository BaseEntries	The **listIdMgr Repository BaseEntries** command lists the base entries for a specified repository.	• Parameters: **- id** The ID of the repository. (String, required) • Returns: A hash map that contains base entry name as key and `nameInRepository` as value.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrRepository BaseEntries {-id id1}` • Using Jython string: `AdminTask.listIdMgrRepository BaseEntries ('[-id id1]')` • Using Jython list: `AdminTask.listIdMgrRepository BaseEntries (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrRepository BaseEntries {-interactive}` • Using Jython string: `AdminTask.listIdMgrRepository BaseEntries ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrRepository BaseEntries (['-interactive'])`
listIdMgr Supported DBTypes	The **listIdMgr Supported DBTypes** command returns a list of supported database types.	• Parameters: None • Returns: A list of supported database types.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrSupportedDBTypes` • Using Jython string: `AdminTask.listIdMgrSupportedDBTypes()` • Using Jython list: `AdminTask.listIdMgrSupportedDBTypes()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrSupported DBTypes {-interactive}` • Using Jython string: `AdminTask.listIdMgrSupported DBTypes ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrSupported DBTypes (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMg rSupported Message Digest Algorithms	The **listIdMgr Supported Message Digest Algorithms** command returns a list of supported message digest algorithms.	• Parameters: None • Returns: A list of supported message digest algorithms.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrSupported MessageDigestAlgorithms` • Using Jython string: `AdminTask.listIdMgrSupported MessageDigestAlgorithms()` • Using Jython list: `AdminTask.listIdMgrSupported MessageDigestAlgorithms()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrSupportedMes sageDigestAlgorithms {-interactive}` • Using Jython string: `AdminTask.listIdMgrSupportedMessage DigestAlgorithms ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrSupportedMessage DigestAlgorithms (['-interactive'])`
listIdMgr Supported LDAPSe rverTypes	The **listIdMgr Supported LDAP ServerTypes** command returns a list of supported LDAP server types.	• Parameters: None • Returns: A list of supported LDAP server types.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrSupported LDAPServerTypes` • Using Jython string: `AdminTask.listIdMgrSupported LDAPServerTypes()` • Using Jython list: `AdminTask.listIdMgrSupported LDAPServerTypes()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrSupported LDAPServerTypes {-interactive}` • Using Jython string: `AdminTask.listIdMgrSupported LDAPServerTypes ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrSupported LDAPServerTypes (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
removeIdMgr LDAPBackupServer	The **removeIdMgr LDAPBackupServer**command removes the backup LDAP server or servers.	• Parameters: **- id** The ID of the repository. (String, required) **- primary_host** The host name for the primary LDAP server. (String, required) **- host** The name of the backup host name. Use a asterisk (*) if you want to remove all backup servers. (String, required) **- port** The port number of the LDAP server. (Integer, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask removeIdMgrLDAPBackupServer {-id id1 -primary_host myprimaryhost.ibm.com -host myhost.ibm.com}` • Using Jython string: `AdminTask.removeIdMgrLDAPBackupServer ('[-id id1 -primary_host myprimaryhost.ibm.com -host myhost.ibm.com]')` • Using Jython list: `AdminTask.removeIdMgrLDAPBackupServer (['-id', 'id1', '-primary_host', 'myprimaryhost.ibm.com', '-host', 'myhost.ibm.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeIdMgrLDAPBackupServer {-interactive}` • Using Jython string: `AdminTask.removeIdMgrLDAPBackupServer ('[-interactive]')` • Using Jython list: `AdminTask.removeIdMgrLDAPBackupServer (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgr Custom Property	The **setIdMgr Custom Property** command adds the custom properties to a repository configuration.	• Parameters: **- id** The ID of the repository. Valid values include the existing repository IDs. (String, required) **- name** The name of the additional property for the repository that are not defined OOTB.(String, required) **- value** If this parameter is an empty string, the property will be deleted from the repository configuration. If this parameter is not an empty string and name does not exist, it will be added. If name is an empty string, all the custom properties will be deleted. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrCustomProperty {-id id1 -name name1 -value value}` • Using Jython string: `AdminTask.setIdMgrCustomProperty ('[-id id1 -name name1 -value value]')` • Using Jython list: `AdminTask.setIdMgrCustomProperty (['-id', 'id1', '-name', 'name1', '-value', 'value'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrCustom Property {-interactive}` • Using Jython string: `AdminTask.setIdMgrCustom Property ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrCustom Property (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgr LDAPA ttrCache	The **setIdMgr LDAPA ttrCache** command configures the LDAP attribute cache configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- cachesDiskOffLoad** (String, optional) **- enabled** Indicates if you want to enable attribute caching. The default value is `true`. (Boolean, optional) **- cacheSize** The maximum size of the attribute cache defined by the number of attribute objects that are permitted in the attribute cache. The minimum value of this parameter is `100`. The default value is `4000`. (Integer, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAPAttrCache {-id id1}` • Using Jython string: `AdminTask.setIdMgrLDAPAttrCache ('[-id id1]')` • Using Jython list: `AdminTask.setIdMgrLDAPAttrCache (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAPAttrCache {-interactive}` • Using Jython string: `AdminTask.setIdMgrLDAPAttrCache ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrLDAPAttrCache (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- cacheTimeOut The amount of time in seconds before the cached entries that are located in the attributes cache can be not valid. The minimum value of this parameter is `0`. The attribute objects that are cached will remain in the attributes cache until the virtual member manager changes the attribute objects. The default value is `1200`. (Integer, optional) **- attributeSizeLimit** An integer that represents the maximum number of attribute object values that can cache in the attributes cache. Some attributes, for example, the member attribute, contain many values. The attributeSizeLimit parameter prevents the attributes cache to cache large attributes. The default value is `2000`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- serverTTLAttribute The name of the ttl attribute that is supported by the LDAP server. The attributes cache uses the value of this attribute to determine when the cached entries in the attributes cache will time out. The ttl attribute contains the time, in seconds, that any information from the entry should be kept by a client before it is considered stale and a new copy is fetched. A value of `0` implies that the object will not be cached. For more information about this attribute, go to: http://www.ietf.org/proceedings/98aug/I-D/draft-ietf-asid-ldap-cache-01.txt. The ttl attribute is not supported by all LDAP servers. If this attribute is supported by an LDAP server, you can set the value of the serverTTLAttribute parameter to the name of the ttl attribute in order to allow the value of the ttyl attribute to determine when cached entries will time out. The time out value for different entries in attributes cache can be different.	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		For example, if the value of the serverTTLAttribute parameter is ttl and the attributes cache retrieves attributes of a user from an LDAP server, it will also retrieve the value of the ttl attribute of this user. If the value is 200, the WMM uses this value to set the time out for the attributes of the user in the attributes cache instead of using the value of cacheTimeout. You can set different ttl attribute values for different users. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgrLDAP ContextPool	The **setIdMgr LDAPContextPool**command sets up the LDAP context pool configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- enabled** By default, the context pool is enabled. If you set this parameter to `false`, the context pool is disabled. When the context pool is disabled, new context instances will be created for each request. The default value is `true`. (Boolean, optional) **- initPoolSize** The number of context instances that the the virtual member manager LDAP adapter creates when it creates the pool. The valid range for this parameter is `1` to `50`. The default value is `1`. (Integer, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAPContextPool {-id id1}` • Using Jython string: `AdminTask.setIdMgrLDAPContextPool ('[-id id1]')` • Using Jython list: `AdminTask.setIdMgrLDAPContextPool (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAPContextPool {-interactive}` • Using Jython string: `AdminTask.setIdMgrLDAPContextPool ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrLDAPContextPool (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- maxPoolSize The maximum number of context instances that the context pool will maintain. Context instances that are in use and those that are idle contribute to this number. When the pool size reaches this number, new context instances cannot be created for new requests. The new request is blocked until a context instance is released by another request or is removed. The request checks periodically if there are context instances available in the pool according to the amount of time that you specify using the poolWaitTime parameter. The minimum value for this parameter is `0`. There is no maximum value. Setting the value of this parameter to `0` means that there is no maximum size and a request for a pooled context instance will use an existing pooled idle context instance or a newly created pooled context instance. The default value is `20`.(Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- prefPoolSize The preferred number of context instances that the context pool will maintain. Context instances that are in use and those that are idle contribute to this number. When there is a request for the use of a pooled context instance and the pool size is less than the preferred size, the context pool creates and uses a new pooled context instance regardless of whether an idle connection is available. When a request finishes with a pooled context instance and the pool size is greater than the preferred size, the context pool closes and removes the pooled context instance from the pool. The valid range for this parameter is from `0` to `100`. Setting the value of this parameter to `0` means that there is no preferred size and a request for a pooled context instance results in a newly created context instance only if no idle ones are available. The default value is `3`.(Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- poolTimeOut An integer that represents the number of milliseconds that an idle context instance may remain in the pool without being closed and removed from the pool. When a context instance is requested from the pool, if this context already exists in the pool for more than the time defined by poolTimeout, this connection will be closed no matter this context instance is stale or active. A new context instance will be created and put back to the pool after it has been released from the request. The minimum value for this parameter is `0`. There is no maximum value. Setting the value of this parameter to `0` means that the context instances in the pool will remain in the pool until they are staled. The context pool catches the communication exception and recreates a new context instance. The default value is `0`.(Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- poolWaitTime The time interval in milliseconds that the request waits until the context pool rechecks if there are idle context instances available in the pool when the number of context instances reaches the maximum pool size. If no idle context instance, the request will continue waiting for the same period of time until next checking. The minimum value for the poolWaitout parameter is `0`. There is no maximum value. A value of `0` for this parameter means that the context pool will not check if idle context exists. The request will be notified when a context instance releases from other requests. The default value is `3000`.(Integer, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgr LDAPG roupConfig	The **setIdMgr LDAPGr oupConfig** command sets up the LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- updateGroup Membership** Updates the group membership if the member is deleted or renamed. Some LDAP servers, for example, Domino server, do not clean up the membership of the user when a user is deleted or renamed. If you choose these LDAP server types in the ldapServerType property, the value of this parameter is set to `true`. Use this parameter to change the value. The default value is `false`. (Boolean, optional) **- name** The name of the membership attribute. For example, `memberOf` in an active directory server and `ibm-allGroups` in IDS. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAP GroupConfig {-id id1}` • Using Jython string: `AdminTask.setIdMgrLDAP GroupConfig ('[-id id1]')` • Using Jython list: `AdminTask.setIdMgrLDAP GroupConfig (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAPGr oupConfig {-interactive}` • Using Jython string: `AdminTask.setIdMgrLDAPGr oupConfig ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrLDAPGroup Config (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- scope The scope of the membership attribute. The following are the possible values for this parameter: • direct - The membership attribute only contains direct groups. Direct groups contain the member and are not contained through a nested group. For example, if group1 contains group2, group2 contains user1, then group2 is a direct group of user1, but group1 is not a direct group of user1. • nested - The membership attribute contains both direct groups and nested groups. • all - The membership attribute contains direct groups, nested groups, and dynamic members. The default value is `direct`. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgr LDAPSearch ResultCache	The **setIdMgr LDAPSearch ResultCache** command sets up the LDAP search result cache configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- cachesDiskOffLoad** Loads the attributes caches and the search results onto hard disk. By default, when the number of cache entries reaches the maximum size of the cache, cache entries are evicted to allow new entries to enter the caches. If you enable this parameter, the evicted cache entries will be copied to disk for future access. The default value is `false`. (Boolean, optional) **- enabled** Enables the search results cache. The default value is `true`. (Boolean, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAPSearch ResultCache {-id id1}` • Using Jython string: `AdminTask.setIdMgrLDAPSearch ResultCache ('[-id id1]')` • Using Jython list: `AdminTask.setIdMgrLDAPSearch ResultCache (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrLDAPSearch ResultCache {-interactive}` • Using Jython string: `AdminTask.setIdMgrLDAPSearch ResultCache ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrLDAPSearch ResultCache (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- cacheSize The maximum size of the search results cache. The number of naming enumeration objects that can be put into the search results cache. The minimum value of this parameter is `100`. The default value is `2000`. (Integer, optional) **- cacheTimeOut** The amount of time in seconds before the cached entries in the search results cache can be not valid. The minimum value for this parameter is `0`. A value of `0` means that the cached naming enumeration objects will stay in the search results cache until there are configuration changes. The default value is `600`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		-searchResultSizeLimit The maximum number of entries contained in the naming enumeration object that can be cached in the search results cache.For example, if the results from a search contains 2000 users, the search results will not cache in the search results cache if the value of the of this property is set to `1000`. The default value is `1000`. (Integer, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgr Entry Mapping Repository	The **setIdMgr Entry Mapping Repository** command sets or updates an entry mapping repository configuration.	• Parameters: **- dataSourceName** The name of the data source. The default value is `jdbc/wimDS`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) **- databaseType** The type of the database. The default value is `DB2`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) **- dbURL** The URL of the database. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrEntry MappingRepository {-dbAdminId dbid1 -dbAdminPassword pw1}` • Using Jython string: `AdminTask.setIdMgrEntry MappingRepository ('[-dbAdminId dbid1 -dbAdminPassword pw1]')` • Using Jython list: `AdminTask.setIdMgrEntry MappingRepository (['-dbAdminId', 'dbid1', '-dbAdmin Password', 'pw1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrEntryMapping Repository {-interactive}` • Using Jython string: `AdminTask.setIdMgrEntryMapping Repository ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrEntryMapping Repository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- dbAdminId The database administrator ID. (String, required if database type is not Cloudscape.) **- dbAdminPassword** The database administrator password. (String, required if database type is not Cloudscape.) **- JDBCDriverClass** The JDBC driver class name. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgr Property Extension Repository	The **setIdMgr Property Extension Repository** command sets or updates the property extension repository configuration.	• Parameters: **- dataSourceName** The name of the data source. The default value is `jdbc/wimDS`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) **- databaseType** The type of the database. The default value is `DB2`. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) **- dbURL** The URL of the database. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String)	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrProperty ExtensionRepository {-entity RetrievalLimit 10 -JDBC DriverClass classname}` • Using Jython string: `AdminTask.setIdMgrProperty ExtensionRepository ('[-entity RetrievalLimit 10 -JDBC DriverClass classname]')` • Using Jython list: `AdminTask.setIdMgrProperty ExtensionRepository (['-entity RetrievalLimit', '10', '-JDBCDriverClass', 'classname'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrProperty ExtensionRepository {-interactive}` • Using Jython string: `AdminTask.setIdMgrProperty ExtensionRepository ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrPropertyExtensionRepository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- dbAdminId The database administrator ID. (String, required if database type is not Cloudscape.) **- dbAdminPassword** The database administrator password. (String, required if database type is not Cloudscape.) **- entityRetrievalLimit** The limit for the retrieval of entities. (Integer, required) **- JDBCDriverClass** The JDBC driver class name. (String, required) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateId MgrDB Repository	The **updateId MgrDB Repository** command updates the configuration for the database repository that you specify.	• Parameters: **- id** The ID of the repository. (String, required) **- dataSourceName** The name of the data source. The default value is `jdbc/wimDS`. (String, optional) **- databaseType** The type of the database. The default value is `DB2`. (String, optional) **- dbURL** The URL of the database. (String, optional) **- dbAdminId** The database administrator ID. (String, optional) **- dbAdminPassword** The database administrator password. (String, optional) **- entityRetrievalLimit** Indicates the value of the retrieval limit on database entries. The default value is `200`. (Integer, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrDB Repository {-id id1}` • Using Jython string: `AdminTask.updateIdMgrDB Repository ('[-id id1]')` • Using Jython list: `AdminTask.updateIdMgrDB Repository (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrDB Repository {-interactive}` • Using Jython string: `AdminTask.updateIdMgrDB Repository ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrDB Repository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- JDBCDriverClass The JDBC driver class name. (String, optional) **- saltLength** The salt length in bits. The default value is `12`. (Integer, optional) **- encryptionKey** The default value is `rZ15ws0ely9yHk3zCs3sTMv/ho8fY17s`. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateId MgrFile Repository	The **updateId MgrFile Repository** command updates the configuration for the file repository that you specify. To update other properties of the file repository use the **update IdMgr Repository**command.	• Parameters: **- id** The ID of the repository. (String, required) **- messageDigest Algorithm** The message digest algorithm that will be used for hashing the password. The default value is `SHA-1`. Valid values include the following: `SHA-245`, `SHA-384`, or `SHA-512`.(String, optional) **- baseDirectory** The base directory where the fill will be created in order to store the data. The default is to be dynamically built during run time using user.install.root and cell name. (String, optional) **- fileName** The file name of the repository. The default value is `fileRegistry.xml`. (String, optional) **- saltLength** The salt length of the randomly generated salt for password hashing. The default value is `12`. (Integer, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrFile Repository {-id id1}` • Using Jython string: `AdminTask.updateIdMgrFile Repository ('[-id id1]')` • Using Jython list: `AdminTask.updateIdMgrFile Repository (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrFile Repository {-interactive}` • Using Jython string: `AdminTask.updateIdMgrFile Repository ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrFile Repository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateId MgrLDAP AttrCache	The **updateId MgrLDAP AttrCache** command updates the LDAP attribute cache configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- cachesDiskOffLoad** (String, optional) **- enabled** Indicates if you want to enable attribute caching. The default value is `true`. (Boolean, optional) **- cacheSize** The maximum size of the attribute cache defined by the number of attribute objects that are permitted in the attribute cache. The minimum value of this parameter is `100`. The default value is `4000`. (Integer, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP AttrCache {-id id1}` • Using Jython string: `AdminTask.updateIdMgrLDAP AttrCache ('[-id id1]')` • Using Jython list: `AdminTask.updateIdMgrLDAP AttrCache (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP AttrCache {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAP AttrCache ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAP AttrCache (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- cacheTimeOut The amount of time in seconds before the cached entries that are located in the attributes cache can be not valid. The minimum value of this parameter is `0`. The attribute objects that are cached will remain in the attributes cache until the virtual member manager changes the attribute objects. The default value is `1200`. (Integer, optional) **- attributeSizeLimit** An integer that represents the maximum number of attribute object values that can cache in the attributes cache. Some attributes, for example, the member attribute, contain many values. The attributeSizeLimit parameter prevents the attributes cache to cache large attributes. The default value is `2000`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- serverTTLAttribute The name of the ttl attribute that is supported by the LDAP server. The attributes cache uses the value of this attribute to determine when the cached entries in the attributes cache will time out. The ttl attribute contains the time, in seconds, that any information from the entry should be kept by a client before it is considered stale and a new copy is fetched. A value of `0` implies that the object will not be cached. For more information about this attribute, go to: http://www.ietf.org/proceedings/98aug/I-D/draft-ietf-asid-ldap-cache-01.txt. The ttl attribute is not supported by all LDAP servers. If this attribute is supported by an LDAP server, you can set the value of the serverTTLAttribute parameter to the name of the ttl attribute in order to allow the value of the ttyl attribute to determine when cached entries will time out. The time out value for different entries in attributes cache can be different.	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		For example, if the value of the serverTTLAttribute parameter is ttl and the attributes cache retrieves attributes of a user from an LDAP server, it will also retrieve the value of the ttl attribute of this user. If the value is 200, the WMM uses this value to set the time out for the attributes of the user in the attributes cache instead of using the value of cacheTimeout. You can set different ttl attribute values for different users. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateId MgrLDAP ContextPool	The **updateId MgrLDAP ContextPool** command updates the LDAP context pool configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- enabled** By default, the context pool is enabled. If you set the value of this parameter to `false`, the context pool is disabled which means that a new context instance will be created for each request. The default value is `true`. (Boolean, optional) **- initPoolSize** The number of context instances that the virtual member manager LDAP adapter creates when it creates the pool. The valid range for this parameter is `1` to `50`. The default value is `1`. (Integer, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP ContextPool {-id id1}` • Using Jython string: `AdminTask.updateIdMgrLDAP ContextPool ('[-id id1]')` • Using Jython list: `AdminTask.updateIdMgrLDAP ContextPool (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP ContextPool {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAP ContextPool ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAP ContextPool (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- maxPoolSize The maximum number of context instances that can be maintained concurrently by the context pool. Both in-use and idle context instances contribute to this number. When the pool size reaches this number, new context instances cannot created for new request. The new request is blocked until a context instance is released by another request or is removed. The request checks periodically if there are context instances available in the pool according to the value defined for the poolWaitTime parameter. The minimum value of the maxPoolSize parameter is `0`. There is no maximum value. A maximum pool size of 0 means that there is no maximum size and that a request for a pooled context instance will use an existing pooled idle context instance or a newly created pooled context instance. The default value is `20`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- prefPoolSize The preferred number of context instances that the Context Pool should maintain. Both in-use and idle context instances contribute to this number. When there is a request for the use of a pooled context instance and the pool size is less than the preferred size, Context Pool will create and use a new pooled context instance regardless of whether an idle connection is available. When a request is finished with a pooled context instance and the pool size is greater than the preferred size, the Context Pool will close and remove the pooled context instance from the pool. The valid range of the prefPoolSize parameter is `0` to `100`. A preferred pool size of 0 means that there is no preferred size: A request for a pooled context instance will result in a newly created context instance only if no idle ones are available. The default value is `3`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- poolTimeOut An integer that represents the number of milliseconds that an idle context instance may remain in the pool without being closed and removed from the pool. When a context instance is requested from the pool, if this context already exists in the pool for more than the time defined by poolTimeout, this connection will be closed no matter this context instance is stale or active. A new context instance will be created and put back to the pool after it has been released from the request.The minimum value of poolTimeout is 0. There is no maximum value.A poolTimeout of 0 means that the context instances in the pool will remain in the pool until they are staled. In this case, Context Pool will catch the communication exception and recreate a new context instance. The default value is `0`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- poolWaitTime The time interval (in milliseconds) that the request will wait until the Context Pool checks again if there are idle context instance available in the pool when the number of context instances reaches the maximum pool size. If there is still no idle context instance, the request will continue waiting for the same period of time until next checking. The minimum value of poolWaitout is 0. There is no maximum value. A poolWaitTime of 0 means the Context Pool will not check if there are idle context. Instead, the request will be notified when there is a context instance is released from other requests. The default value is `3000`. (Integer, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateId MgrLDAP EntityType	The **updateId MgrLDAP EntityType** command updates an existing LDAP entity type definition to LDAP repository configuration. You can use this command to add more values to multi-valued parameters. If the property already exists, the value of the property will be replaced. If the property does not exist, it will be added.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the entity type. (String, required) **- searchFilter** The search filter that you want to use to search the entity type. (String, optional) **- objectClasses** One or more object classes for the entity type. (String, optional) **- objectClassesForCreate** The object class that will be when you create an entity type object. You do not have to specify the value of this parameter if it is the same as the value of the objectClasses parameter. (String, optional) **- searchBases** The search base or bases to use while searching the entity type. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAPEntity Type {-id id1 -name name1}` • Using Jython string: `AdminTask.updateIdMgrLDAPEntity Type ('[-id id1 -name name1]')` • Using Jython list: `AdminTask.updateIdMgrLDAPEntity Type (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP EntityType {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAP EntityType ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAP EntityType (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr LDAPGroup Dynamic MemberAttr	The **updateIdMgr LDAPGroup Dynamic MemberAttr** command updates a dynamic member attribute configuration to an LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the LDAP attribute that is used as the group member attribute. For example, `memberURL`. (String, required) **- objectClass** The group object class that contains the dynamic member attribute. For example `groupOfURLs`. If you do not define this parameter, the dynamic member attribute will apply to all group object classes. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAPGroup DynamicMemberAttr {-id id1 -name name1 -objectClass groupOfURLs}` • Using Jython string: `AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr ('[-id id1 -name name1 -objectClass groupOfURLs]')` • Using Jython list: `AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-id', 'id1', '-name', 'name1', '-objectClass', 'groupOfURLs'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAPGroup DynamicMemberAttr {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr LDAPGroup MemberAttr	The **updateIdMgr LDAPGroup MemberAttr** command updates a member attribute configuration of an LDAP group configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The name of the LDAP attribute that is used as the group member attribute. For example, `member` or `uniqueMember`. (String, required) **- objectClass** The group object class that contains the member attribute. For example, `groupOfNames` or `groupOfUnqiueNames`. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP GroupMemberAttr {-id id1 -name name1}` • Using Jython string: `AdminTask.updateIdMgrLDAP GroupMemberAttr ('[-id id1 -name name1]')` • Using Jython list: `AdminTask.updateIdMgrLDAP GroupMemberAttr (['-id', 'id1', '-name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAPGroup MemberAttr {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAPGroup MemberAttr ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAPGroup MemberAttr (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- scope The scope of the member attribute. The following are the valid values: • direct - The member attribute only contains direct members whereby the member is directly contained by the group and not contained in a nested group. For example, if group1 contains group2, group2 contains user1, then group2 is a direct member of group1 but user1 is not a direct member of group1. Both member and uniqueMember are direct member attributes. • nested - The member attribute contains both direct members and nested members.	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		• all - The member attribute contains direct members, nested members and dynamic members. One example is the ibm-allMembers attribute that IBM Tivoli Directory Server supports. The default value is `direct`. (String, optional) **- dummyMember** When you create a group without specifying a member, a dummy member will be filled in automatically to avoid receiving an exception that indicates that there is a mandatory attribute missing. (String, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateId MgrLDAP Repository	The **updateId MgrLDAP Repository** command updates an LDAP repository configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- ldapServerType** The type of LDAP server that is being used. The default value is `IDS51`. (String, optional) **- adapterClassName** The default value is `com.ibm.ws.wim.adapter.ldap.LdapAdapter`. (String, optional) **- certificateMapMode** Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is `exactdn`. To use the certificate filter for the mapping, specify `certificatefilter`. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP Repository {-id id1}` • Using Jython string: `AdminTask.updateIdMgrLDAP Repository ('[-id id1]')` • Using Jython list: `AdminTask.updateIdMgrLDAP Repository (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP Repository {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAP Repository ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAP Repository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- certificateFilter If `certificateMapMode` has the value `certificatefilter`, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional) **- isExtIdUnique** Specifies if the external ID is unique. The default value is `true`. (Boolean, optional) **- loginProperties** Indicates the property name used for login. (String , optional) **- primaryServerQueryTimeInterval** Indicates the polling interval for testing the primary server availability. The value of this parameter is specified in minutes. The default value is `15`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- returnToPrimaryServer Indicates to return to the primary LDAP server when it is available. The default value is `true`. (Boolean, optional) **- supportAsyncMode** Indicates if the async mode is supported or not. The default value is `false`. (Boolean, optional) **- supportSorting** Indicates if sorting is supported or not. The default value is `false`. (Boolean, optional) **- supportPaging** Indicates if paging is supported or not. The default value is `false`. (Boolean, optional) **- supportTransaction** Indicates if transactions are supported or not. The default value is `false`. (Boolean, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- supportExternalName Indicates if external names are supported or not. The default value is `false`. (Boolean, optional) **- sslConfiguration** The SSL configuration. (String, optional) **- translateRDN** Indicates to translate RDN or not. The default value is `false`. (Boolean, optional) **- searchTimeLimit** The value of search time limit. (Integer, optional) **- searchCountLimit** The value of search count limit. (Integer, optional) **- searchPageSize** The value of search page size. (Integer, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr LDAPSearch ResultCache	The **updateIdMgr LDAPSearch ResultCache** command updates the LDAP search result cache configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- cachesDiskOffLoad** Loads the attributes caches and the search results onto hard disk. By default, when the number of cache entries reaches the maximum size of the cache, cache entries are evicted to allow new entries to enter the caches. If you enable this parameter, the evicted cache entries will be copied to disk for future access. The default value is `false`. (Boolean, optional) **- enabled** Enables the search results cache. The default value is `true`. (Boolean, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP SearchResultCache {-id id1}` • Using Jython string: `AdminTask.updateIdMgrLDAPSearch ResultCache ('[-id id1]')` • Using Jython list: `AdminTask.updateIdMgrLDAPSearch ResultCache (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP SearchResultCache {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAPSearch ResultCache ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAPSearch ResultCache (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- cacheSize The maximum size of the search results cache. The number of naming enumeration objects that can be put into the search results cache. The minimum value of this parameter is `100`. The default value is `2000`. (Integer, optional) **- cacheTimeOut** The amount of time in seconds before the cached entries in the search results cache can be not valid. The minimum value for this parameter is `0`. A value of `0` means that the cached naming enumeration objects will stay in the search results cache until there are configuration changes. The default value is `600`. (Integer, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		-searchResultSizeLimit The maximum number of entries contained in the naming enumeration object that can be cached in the search results cache.For example, if the results from a search contains 2000 users, the search results will not cache in the search results cache if the value of the of this property is set to `1000`. The default value is `1000`. (Integer, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr LDAPServer	The **updateIdMgr LDAPServer**command updates an LDAP server configuration for the LDAP repository ID that you specify.	• Parameters: **- id** The ID of the repository. (String, required) **- host** The host name for the LDAP server that contains the properties that you want to modify. (String, required) **- port** The port number for the LDAP server. (Integer, optional) **- authentication** Indicates the authentication method to use. The default value is `simple`. Valid values include: `none` or `strong`. (String, optional) **- bindDN** The binding domain name for the LDAP server. (String, optional) **- bindPassword** The binding password. The password is encrypted before it is stored.(String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAPServer {-id id1 -host myhost.ibm.com}` • Using Jython string: `AdminTask.updateIdMgrLDAPServer ('[-id id1 -host myhost.ibm.com]')` • Using Jython list: `AdminTask.updateIdMgrLDAPServer (['-id', 'id1', '-host', 'myhost.ibm.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrLDAP Server {-interactive}` • Using Jython string: `AdminTask.updateIdMgrLDAP Server ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrLDAP Server (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- certificateMapMode Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is `exactdn`. To use the certificate filter for the mapping, specify `certificatefilter`. (String, optional) **- certificateFilter** If `certificateMapMode` has the value `certificatefilter`, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional) **- connectTimeout** The connection timeout measured in seconds. The default value is `0`. (Integer, optional) **- connectionPool** The connection pool. The default value is `false`. (Boolean, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- derefAliases Controls how aliases are dereferenced. The default value is `always`. Valid values include: • `never` - never deference aliases • `finding` - deferences aliases only during name resolution • `searching` - deferences aliases only after name resolution (String, optional) **- ldapServerType** The type of LDAP server being used. The default value is `IDS51`. (String, optional) **- primary_host** The host name for the primary LDAP server. (String, optional) **- referal** The LDAP referral. The default value is `ignore`. Valid values include: `follow`, `throw`, or `false`. (String, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- sslConfiguration The SSL configuration. (String, optional) **- sslEnabled** Indicates to enable SSL or not. The default value is `false`. (Boolean, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr Repository	The **updateIdMgr Repository**command updates the common repository configuration.	• Parameters: **- id** The ID of the repository. (String, required) **- adapterClassName** The implementation class name for the repository adapter. (String, optional) **- EntityTypesNotAllowCreate** The name of the entity type that should not be created in this repository. (String, optional) **- EntityTypesNotAllowUpdate** The name of the entity type that should not be updated in this repository. (String, optional) **- EntityTypesNotAllowRead** The name of the entity type that should not be read from this repository. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrRepository {-id id1}` • Using Jython string: `AdminTask.updateIdMgrRepository ('[-id id1]')` • Using Jython list: `AdminTask.updateIdMgrRepository (['-id', 'id1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrRepository {-interactive}` • Using Jython string: `AdminTask.updateIdMgrRepository ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrRepository (['-interactive'])`

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- EntityTypesNotAllowDelete The name of the entity type that should not be deleted from this repository. (String, optional) **- loginProperties** (String, optional) **- readOnly** Indicates if this is a read only repository. The default value is `false`. (Boolean, optional) **- repositoriesForGroups** The repository ID where group data is stored. (String, optional) **- supportPaging** Indicates if the repository supports paging or not. (Boolean, optional) **- supportSorting** Indicates if the repository supports sorting or not. (Boolean, optional)	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- supportTransaction Indicates if the repository supports transaction or not. (Boolean, optional) **- isExtIdUnique** Specifies if the external ID is unique or not. (Boolean, optional) **- supportedExternalName** Indicates if the repository supports external names or not. (Boolean, optional) **- supportAsyncMode** Indicates if the adapter supports async mode or not. The default value is `false`. (Boolean, optional) • Returns: None	

Table 25. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr Repository BaseEntry	The **updateIdMgr Repository BaseEntry** command updates a base entry to the specified repository.	• Parameters: **- id** The ID of the repository. (String, required) **- name** The distinguished name of a base entry. (String, required) **- nameInRepository** The distinguished name in the repository that uniquely identifies the base entry name. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrReposi toryBaseEntry {-id id1 name name1}` • Using Jython string: `AdminTask.updateIdMgrRepository BaseEntry ('[-id id1 name name1]')` • Using Jython list: `AdminTask.updateIdMgrReposi toryBaseEntry (['-id', 'id1', 'name', 'name1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrReposi toryBaseEntry {-interactive}` • Using Jython string: `AdminTask.updateIdMgrReposi toryBaseEntry ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrReposi toryBaseEntry (['-interactive'])`

Commands for the IdMgrRealmConfig group of the AdminTask object

Use the commands in the IdMgrConfig group to configure the member manager realm and realms. The commands for this group do not require a target object. To see the additional commands related to the member manager, see the `Commands for the IdMgrRepositoryConfig group of the AdminTask object and the Commands for the IdMgrConfig group of the AdminTask object articles.

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the IdMgrRealmConfig group of the AdminTask object:

Table 26.

Command name:	Description:	Parameters and return values:	Examples:
addIdMgrRealmBaseEntry	The **addIdMgrRealmBaseEntry** command adds a base entry to a specified realm configuration.	• Parameters: **- name** The realm name. (String, required) **- baseEntry** The name of a base entry. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask addIdMgrRealmBaseEntry {-name realm1 -baseEntry entry1}` • Using Jython string: `AdminTask addIdMgrRealmBaseEntry ('[-name realm1 -baseEntry entry1]')` • Using Jython list: `AdminTask addIdMgrRealmBaseEntry (['-name', 'realm1', '-base Entry', 'entry1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addIdMgrRealm BaseEntry {-interactive}` • Using Jython string: `AdminTask.addIdMgrRealmBase Entry ('[-interactive]')` • Using Jython list: `AdminTask.addIdMgrRealmBase Entry (['-interactive'])`

Table 26. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createIdMgrRealm	The **createIdMgrRealm** command creates a realm configuration.	• Parameters: **- name** The realm name. (String, required) **- securityUse** A string that indicates if this virtual realm will be used in security now, later, or never. The default value is `active`. Additional values includes: `inactive` and `nonSelectable`. (String, optional) **- delimiter** The delimiter used for this realm. The default value is `@`. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask createIdMgrRealm {-name realm1}` • Using Jython string: `AdminTask.createIdMgrRealm ('[-name realm1]')` • Using Jython list: `AdminTask.createIdMgrRealm (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createIdMgrRealm {-interactive}` • Using Jython string: `AdminTask.createIdMgrRealm ('[-interactive]')` • Using Jython list: `AdminTask.createIdMgrRealm (['-interactive'])`
deleteIdMgrRealm	The **deleteIdMgrRealm** command deletes the realm configuration that you specified.	• Parameters: **- name** The realm name. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrRealm {-name realm1}` • Using Jython string: `AdminTask.deleteIdMgrRealm ('[-name realm1]')` • Using Jython list: `AdminTask.deleteIdMgrRealm (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrRealm {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrRealm ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrRealm (['-interactive'])`

Table 26. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteIdMgr RealmBaseEntry	The **deleteIdMgr RealmBaseEntry** command deletes a base entry from a realm configuration that you specified. The realm must always contain at least one base entry, thus you cannot remove every entry.	• Parameters: **- name** The realm name. (String, required) **- baseEntry** The name of a base entry. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrRealmBaseEntry {-name realm1 -baseEntry entry1}` • Using Jython string: `AdminTask.deleteIdMgrRealmBaseEntry ('[-name realm1 -baseEntry entry1]')` • Using Jython list: `AdminTask.deleteIdMgrRealmBaseEntry (['-name', 'realm1', '-base Entry', 'entry1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteIdMgrRealm BaseEntry {-interactive}` • Using Jython string: `AdminTask.deleteIdMgrRealm BaseEntry ('[-interactive]')` • Using Jython list: `AdminTask.deleteIdMgrRealm BaseEntry (['-interactive'])`
getIdMgrDefaultRealm	The **getIdMgrDefaultRealm** command returns the default realm name.	• Parameters: None • Returns: The name of the default realm.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrDefaultRealm` • Using Jython string: `AdminTask.getIdMgrDefaultRealm()` • Using Jython list: `AdminTask.getIdMgrDefaultRealm()` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrDefault Realm {-interactive}` • Using Jython string: `AdminTask.getIdMgrDefault Realm ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrDefault Realm (['-interactive'])`

Table 26. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgrRepositories ForRealm	The **getIdMgrRepositories ForRealm** command returns repository specific details for the repositories configured for a specified realm.	• Parameters: **- name** The realm name. (String, required) • Returns: A hash map with the following keys: – id - The repository ID. – repositoryType - The type of repository, for example, File, LDAP, DB, and so on. – specificRepositoryType - The specific type of repository, for example, for LDAP, IDS51, NDS, and so on. – host - The host name where the repository resides. For example, for File, LocalHost or for DB, dataSourceName. – port - The port number. This only applies to LDAP. – name - The name of the base entry. – nameInRepository - The name in repository for the base entry. This command will not return the property extension and entry mapping repository data.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrRepositories ForRealm {-name realm1}` • Using Jython string: `AdminTask.getIdMgrRepositories ForRealm ('[-name realm1]')` • Using Jython list: `AdminTask.getIdMgrRepositories ForRealm (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrRepositories ForRealm {-interactive}` • Using Jython string: `AdminTask.getIdMgrRepositories ForRealm ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrRepositories ForRealm (['-interactive'])`

Table 26. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgrRealm	The **getIdMgrRealm** command returns the configuration parameters for the realm that you specified.	• Parameters: **- name** The realm name. (String, required) • Returns: A hash map that contains keys as the parameters of the **createIdMgrRealm** command.	**Batch mode example usage:** • Using Jacl: `$AdminTask getIdMgrRealm {-name realm1}` • Using Jython string: `AdminTask.getIdMgrRealm ('[-name realm1]')` • Using Jython list: `AdminTask.getIdMgrRealm (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getIdMgrRealm {-interactive}` • Using Jython string: `AdminTask.getIdMgrRealm ('[-interactive]')` • Using Jython list: `AdminTask.getIdMgrRealm (['-interactive'])`
listIdMgrRealms	The **listIdMgrRealms** command returns all of the names of the configured realms.	• Parameters: None • Returns: A list that contains the name of the configured realms.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrRealms` • Using Jython string: `AdminTask.listIdMgrRealms()` • Using Jython list: `AdminTask.listIdMgrRealms()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrRealms {-interactive}` • Using Jython string: `AdminTask.listIdMgrRealms ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrRealms (['-interactive'])`

Table 26. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgrRealm BaseEntries	The **listIdMgrRealm BaseEntries** command returns all of the names of the configured realms.	• Parameters: **- name** The realm name. (String, required) • Returns: A list of all the base entries of the specified realm.	**Batch mode example usage:** • Using Jacl: `$AdminTask listIdMgrRealmBase Entries {-name realm1}` • Using Jython string: `AdminTask.listIdMgrRealmBase Entries ('[-name realm1]')` • Using Jython list: `AdminTask.listIdMgrRealmBase Entries (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listIdMgrRealmBase Entries {-interactive}` • Using Jython string: `AdminTask.listIdMgrRealmBase Entries ('[-interactive]')` • Using Jython list: `AdminTask.listIdMgrRealmBase Entries (['-interactive'])`
renameIdMgrRealm	The **renameIdMgrRealm** command renames the name of the realm that you specified.	• Parameters: **- name** The realm name. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask renameIdMgrRealm {-name realm1}` • Using Jython string: `AdminTask.renameIdMgrRealm ('[-name realm1]')` • Using Jython list: `AdminTask.renameIdMgrRealm (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask renameIdMgrRealm {-interactive}` • Using Jython string: `AdminTask.renameIdMgrRealm ('[-interactive]')` • Using Jython list: `AdminTask.renameIdMgrRealm (['-interactive'])`

Table 26. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgrDefaultRealm	The **setIdMgrDefaultRealm** command sets up the default realm configuration.	• Parameters: **- name** The name of the realm that is used as a default realm when the caller does not specify any in context. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask setIdMgrDefaultRealm {-name realm1}` • Using Jython string: `AdminTask.setIdMgrDefaultRealm ('[-name realm1]')` • Using Jython list: `AdminTask.setIdMgrDefaultRealm (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask setIdMgrDefault Realm {-interactive}` • Using Jython string: `AdminTask.setIdMgrDefault Realm ('[-interactive]')` • Using Jython list: `AdminTask.setIdMgrDefault Realm (['-interactive'])`

Table 26. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgrRealm	The **updateIdMgrRealm** command updates the configuration for a realm that you specify.	• Parameters: **- name** The realm name. (String, required) **- securityUse** A string that indicates if this realm will be used in security now, later, or never. The default value is `active`. Additional values includes: `inactive` and `nonSelectable`. (String, optional) **- delimiter** The delimiter used for this realm. The default value is `@`. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask updateIdMgrRealm {-name realm1}` • Using Jython string: `AdminTask.updateIdMgrRealm ('[-name realm1]')` • Using Jython list: `AdminTask.updateIdMgrRealm (['-name', 'realm1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateIdMgrRealm {-interactive}` • Using Jython string: `AdminTask.updateIdMgrRealm ('[-interactive]')` • Using Jython list: `AdminTask.updateIdMgrRealm (['-interactive'])`

Commands for the WIMManagementCommands group of the AdminTask object

Use the commands in the WIMManagementCommands group to configure the virtual member manager. The commands for this group do not require a target object. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the WIMManagementCommands group of the AdminTask object:

Table 27.

Command name:	Description:	Parameters and return values:	Examples:
addMemberToGroup	The **addMemberToGroup** command adds a user or a group to a group.	• Parameters: **- memberUniqueName** Specifies the unique name value for the user or group that you want to add to the specified group. This parameter maps to the uniqueName property in the virtual member manager. (String, required) **- groupUniqueName** Specifies the unique name value for the group to which you want to add the user or group that you specified in the memberUniqueName parameter. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails.	**Batch mode example usage:** • Using Jacl: `$AdminTask addMemberToGroup {-memberUniqueName uid=meyersd,cn=users,dc=yourco,dc=com -groupUniqueName cn=admins,cn=groups,dc=yourco,dc=com}` • Using Jython string: `AdminTask.addMemberToGroup ('[-memberUniqueName uid=meyersd,cn=users,dc=yourco,dc=com -group UniqueName cn=admins,cn=groups,dc=yourco,dc=com]')` • Using Jython list: `AdminTask.addMemberToGroup (['-memberUniqueName', 'uid=meyersd,cn=users,dc=yourco,dc=com', '-group UniqueName', 'cn=admins,cn=groups,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addMemberToGroup {-interactive}` • Using Jython string: `AdminTask.addMemberToGroup ('[-interactive]')` • Using Jython list: `AdminTask.addMemberToGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createGroup	The **createGroup** command creates a new group in the virtual member manager. After the command completes, the new group will appear in the repository. For LDAP, a group must contain a member. The memberUniqueName parameter is optional in this case. If you set the memberUniqueName parameter to the unique name of a group or a user, the group or user will be added as a member of the group.	• Parameters: **- cn** Specifies the common name for the group that you want to create. This parameter maps to the cn property in virtual member manager. (String, required) **- description** Specifies additional information about the group that you want to create. This parameter maps to the description property in a virtual member manager object. (String, optional) **- parent** Specifies the repository in which you want to create the group. This parameter maps to the parent property in the virtual member manager. (String, optional) **- memberUniqueName** Specifies the unique name value for the user or group that you want to add to the new group. This parameter maps to the uniqueName property in the virtual member manager. (String, optional) • Returns: The unique name of the group that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createGroup {-cn groupA -description a group of admins}` • Using Jython string: `AdminTask.createGroup ('[-cn groupA -description a group of admins]')` • Using Jython list: `AdminTask.createGroup (['-cn', 'groupA', '-description', 'a group of admins'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createGroup {-interactive}` • Using Jython string: `AdminTask.createGroup ('[-interactive]')` • Using Jython list: `AdminTask.createGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createUser	The **createUser** command creates a new user in the default repository or a repository that the parent command parameter specifies. This command creates a person entity and a login account entity in the virtual member manager.	• Parameters: **- uid** Specifies the unique ID for the user that you want to create. Virtual member manager then creates a uniqueId value and a uniqueName value for the user. This parameter maps to the uid property in the virutal member manager. (String, required) **- password** Specifies the password for the user. This parameter maps to the password property in the virtual member manager. (String, required) **- confirmPassword** Specifies the password again to validate how it was entered for the password parameter. This parameter maps to the password property in virtual member manager. (String, optional) **- cn** Specifes the first name or given name of the user. This parameter maps to the cn property in virutal member manager. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createUser {-uid 123 -password tempPass -confirmPassword tempPass -cn Jane -surname Doe -ibm-primaryEmail janedoe@acme.com}` • Using Jython string: `AdminTask.createUser ('[-uid 123 -password tempPass -confirmPassword tempPass -cn Jane -surname Doe -ibm-primaryEmail janedoe@acme.com]')` • Using Jython list: `AdminTask.createUser (['-uid', '123', '-password', 'tempPass', '-confirmPassword', 'tempPass', '-cn', 'Jane', '-surname', 'Doe', '-ibm-primaryEmail', 'janedoe@acme.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createUser {-interactive}` • Using Jython string: `AdminTask.createUser ('[-interactive]')` • Using Jython list: `AdminTask.createUser (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- surname Specifies the last name or family name of the user. This parameter maps to the sn property in virtual member manager. (String, optional) **- ibm-primaryEmail** Specifies the e-mail address of the user. This parameter maps to the ibm-PrimaryEmail property in the virtual member manager. (String, optional) **- parent** Specifies the repository in which you want to create the user. This parameter maps to the parent property in the virtual member manager. (String, optional) • Returns: The unique name of the user that you created.	

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteGroup	The **deleteGroup** command deletes a group in the virtual member manager. You cannot use this command to delete descendants. When this command completes, the group will be deleted from the repository.	• Parameters: **- uniqueName** Specifies the unique name value for the group that you want to delete. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: Void if the command is successful. Returns an error if the command fails.	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteGroup {-uniqueName cn=operators,cn=users,dc=yourco,dc=com}` • Using Jython string: `AdminTask.deleteGroup ('[-uniqueName cn=operators,cn=users,dc=yourco,dc=com]')` • Using Jython list: `AdminTask.deleteGroup (['-uniqueName', 'cn=operators,cn=users,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteGroup {-interactive}` • Using Jython string: `AdminTask.deleteGroup ('[-interactive]')` • Using Jython list: `AdminTask.deleteGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteUser	The **deleteUser** command deletes a user from the virtual member manager. This includes a person object and an account object in the non-merged repositories.	• Parameters: **- uniqueName** Specifies the unique name value for the user that you want to delete. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: Void if the command is successful and an exception if the command fails.	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteUser {-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com}` • Using Jython string: `AdminTask.deleteUser ('[-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com]')` • Using Jython list: `AdminTask.deleteUser (['-uniqueName', 'uid=dmeyers,cn=users,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteUser {-interactive}` • Using Jython string: `AdminTask.deleteUser ('[-interactive]')` • Using Jython list: `AdminTask.deleteUser (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
duplicate Membership OfGroup	Use the **duplicate Membership OfGroup** command to make a one group a member of all of the same groups as another group. For example, group A is in group B and group C. To add group D to the same groups as group A, use the **duplicate Membership OfGroup** command.	• Parameters: **- copyToName** Specifies the name of the group to which you want to add the memberships of the group specified in the copyFromName parameter. (String, required) **- copyFromName** Specifies the name of the group from which you want to copy the group memberships for another group to use. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails.	**Batch mode example usage:** • Using Jacl: `$AdminTask duplicateMembership OfGroup {-copyToName cn=operators,cn=groups,dc=yourco,dc=com -copyFromName cn=admins,cn=groups,dc=yourco,dc=com}` • Using Jython string: `AdminTask.duplicateMembershipOfGroup ('[-copyToName cn=operators,cn=groups,dc=yourco,dc=com -copyFromName cn=admins,cn=groups,dc=yourco,dc=com]')` • Using Jython list: `AdminTask.duplicateMembershipOfGroup (['-copyToName', 'cn=operators,cn=groups,dc=yourco,dc=com', '-copyFromName', 'cn=admins,cn=groups,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask duplicateMembership OfGroup {-interactive}` • Using Jython string: `AdminTask.duplicateMembershipOfGroup ('[-interactive]')` • Using Jython list: `AdminTask.duplicateMembershipOfGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
duplicate Membership OfUser	Use the **duplicate Membership OfUser** command to make a one user a member of all of the same groups as another user. For example, user 1 is in group B and group C. To add user 2 to the same groups as user 1, use the **duplicate Membership OfUser** command.	• Parameters: **- copyToName** Specifies the name of the user to which you want to add the memberships of the user specified in the copyFromName parameter. (String, required) **- copyFromName** Specifies the name of the user from which you want to copy the group memberships for another user to use. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails.	**Batch mode example usage:** • Using Jacl: `$AdminTask duplicateMembership OfUser {-copyToName uid=meyersd,cn=users,dc=yourco,dc=com -copyFromName uid=jhart,cn=users,dc=yourco,dc=com}` • Using Jython string: `AdminTask.duplicateMembership OfUser ('[-copyToName uid=meyersd,cn=users,dc=yourco,dc=com -copyFromName uid=jhart,cn=users,dc=yourco,dc=com]')` • Using Jython list: `AdminTask.duplicateMembership OfUser (['-copyToName', 'uid=meyersd,cn=users,dc=yourco,dc=com', '-copyFromName', 'uid=jhart,cn=users,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask duplicateMembership OfUser {-interactive}` • Using Jython string: `AdminTask.duplicateMembership OfUser ('[-interactive]')` • Using Jython list: `AdminTask.duplicateMembership OfUser (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getGroup	The **getGroup** command retrieves the common name and description of a group.	• Parameters: **- uniqueName** Specifies the unique name value for the group that you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: A map of common name and description properties.	**Batch mode example usage:** • Using Jacl: `$AdminTask getGroup {-uniqueName cn=operators, cn=groups,dc=yourco,dc=com}` • Using Jython string: `AdminTask.getGroup ('[ -uniqueName cn=operators, cn=groups,dc=yourco, dc=com]')` • Using Jython list: `AdminTask.getGroup (['-uniqueName', 'cn=operators,cn=groups,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getGroup {-interactive}` • Using Jython string: `AdminTask.getGroup ('[-interactive]')` • Using Jython list: `AdminTask.getGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getMembership OfGroup	The **getMembership OfGroup** command retrieves the groups of which a group is a member.	• Parameters: **- uniqueName** Specifies the unique name value for the group whose group memberships you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: A list of unique names of each of the groups of which the group is a member.	**Batch mode example usage:** • Using Jacl: `$AdminTask getMebmership OfGroup {-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com}` • Using Jython string: `AdminTask.getMebmership OfGroup ('[-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com]')` • Using Jython list: `AdminTask.getMebmership OfGroup (['-uniqueName', 'uid=dmeyers,cn=users, dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getMembership OfGroup {-interactive}` • Using Jython string: `AdminTask.getMembership OfGroup ('[-interactive]')` • Using Jython list: `AdminTask.getMembership OfGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getMem bership OfUser	The **getMem bership OfUser** command retrieves the groups of which a user is a member.	• Parameters: **- uniqueName** Specifies the unique name value for the user whose group memberships you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: A list of unique names for each group of which the user is a member.	**Batch mode example usage:** • Using Jacl: `$AdminTask getMebmer shipOfUser {-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com}` • Using Jython string: `AdminTask.getMebmership OfUser ('[-uniqueName uid=dmeyers,cn=users, dc=yourco,dc=com]')` • Using Jython list: `AdminTask.getMebmership OfUser (['-uniqueName', 'uid=dmeyers,cn=users ,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getMembership OfUser {-interactive}` • Using Jython string: `AdminTask.getMembership OfUser ('[-interactive]')` • Using Jython list: `AdminTask.getMembership OfUser (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getMembers OfGroup	The **getMembers OfGroup** command retrieves the members of a group.	• Parameters: **- uniqueName** Specifies the unique name value for the group whose members you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: The unique name of each of the members of the group and the type of each member.	**Batch mode example usage:** • Using Jacl: `$AdminTask getMembersOf Group {-uniqueName cn=operators,cn=groups ,dc=yourco,dc=com}` • Using Jython string: `AdminTask.getMembersOf Group ['(-uniqueName cn=operators,cn=groups ,dc=yourco,dc=com]')` • Using Jython list: `AdminTask.getMembersOf Group [('-uniqueName', 'cn=operators,cn=groups ,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getMembersOfGroup {-interactive}` • Using Jython string: `AdminTask.getMembersOfGroup ('[-interactive]')` • Using Jython list: `AdminTask.getMembersOfGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getUser	The **getUser** command retrieves information about a user in the virtual member manager.	• Parameters: **- uniqueName** Specifies the unique name value for the user that you want to view. This parameter maps to the uniqueName property in the virtual member manager. (String, required) • Returns: A map that contains the following properties: uniqueName, cn, sn, uid, and ibm-primaryEmail. These attributes are fixed and you cannot change them.	**Batch mode example usage:** • Using Jacl: `$AdminTask getUser {-user Name uid=dmeyers,cn=us ers,dc=yourco,dc=com}` • Using Jython string: `AdminTask.getUser ('[-use rName uid=dmeyers,cn= users,dc=yourco,dc=com]')` • Using Jython list: `AdminTask.getUser (['-use rName', 'uid=dmeyers,c n=users,dc=yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getUser {-interactive}` • Using Jython string: `AdminTask.getUser ('[-interactive]')` • Using Jython list: `AdminTask.getUser (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
removeMember FromGroup	The **removeMember FromGroup** command removes a user or a group from a group.	• Parameters: **- memberUniqueName** Specifies the unique name value for the user or group that you want to remove from the specified group. This parameter maps to the uniqueName property in virtual member manager. (String, required) **- groupUniqueName** Specifies the unique name value for the group from which you want to remove the user or group that you specified with the memberUniqueName paramter. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails.	**Batch mode example usage:** • Using Jacl: `$AdminTask removeMember FromGroup {-memberUnique Name uid=meyersd,cn= users,dc=yourco,dc=com -groupUniqueName cn= admins,cn-groups,dc= yourco,dc=com}` • Using Jython string: `AdminTask.removeMemberF romGroup ('[-memberUnique Name uid=meyersd,cn= users,dc=yourco,dc=com -groupUniqueName cn=a dmins,cn-groups,dc=your co,dc=com]')` • Using Jython list: `AdminTask.removeMemberFrom Group (['-memberUniqueName', 'uid=meyersd,cn=users, dc=yourco,dc=com', '-groupUniqueName', 'cn=admins,cn-groups,dc= yourco,dc=com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeMember FromGroup {-interactive}` • Using Jython string: `AdminTask.removeMemberFr omGroup ('[-interactive]')` • Using Jython list: `AdminTask.removeMemberFr omGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
searchGroups	Use the **searchGroups** command to find groups in the virtual member manager that match criteria that you provide. For example, you can use the **searchGroups** command to find all of the groups with a common name that begins with `IBM`. You can search for any virtual member manager property because the command is generic.	• Parameters: **- cn** The first name or given name of the user. This parameter maps to the cn property in the virtual member manager. You must set this parameter or the description parameter, but not both. (String, optional) **- description** Specifies information about the group. This parameter maps to the description entity in a virtual member manager object. You must set this parameter or the cn parameter, but not both. (String, optional) **- timeLimit** Specifies the maximum amount of time in milliseconds that the search can run. The default value is no time limit. (String, optional) **- countLimit** Specifies the maximum number of results that you want returned from the search. By default, all groups found in the search are returned. (String, optional) • Returns: A list of unique names of all of the groups that match the search criteria that you provided.	**Batch mode example usage:** • Using Jacl: `$AdminTask searchGroups {cn *IBM*}` • Using Jython string: `AdminTask.searchGroups ('[cn *IBM*]')` • Using Jython list: `AdminTask.searchGroups (['cn', '*IBM*'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask searchGroups {-interactive}` • Using Jython string: `AdminTask.searchGroups ('[-interactive]')` • Using Jython list: `AdminTask.searchGroups (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
searchUsers	Use the **searchUsers** command to find users in the virtual member manager that match criteria that you provide. For example, you can use the **searchUsers** command to find all of the telephone numbers that contain 919. You can search for any virtual member manager property because the command is generic.	• Parameters: **- principalName** Specifies the principal name oft he user that is used as the logon ID for the user in the system. This parameter maps to the principalName property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional) **- uid** Specifies the unique ID value for the user for whom you want to search. This parameter maps to the uid property in virtual member manage. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional) **- cn** Specifies the first name or given name of the user. This parameter maps to the cn property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask searchUsers {-principalName */IBM/US*}` • Using Jython string: `AdminTask.searchUsers ('[-principalName */IBM/US*]')` • Using Jython list: `AdminTask.searchUsers (['-principalName', '*/IBM/US*'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask searchUsers {-interactive}` • Using Jython string: `AdminTask.searchUsers ('[-interactive]')` • Using Jython list: `AdminTask.searchUsers (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- sn Specifies the last name or family name of the user. This parameter maps to the sn property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional) **- ibm-primaryEmail** Specifies the email address of the user. This parameter maps to the ibm-PrimaryEmail property in the virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional) **- timeLimit** Specifies the maximum amount of time in milliseconds that the search can run. The default is not time limit. (String, optional) **- countLimit** Specifies the maximum number of results that you want returned from the search. By default, all users found int he search are returned. (String, optional) • Returns: A list of unique names of all of the users that match the search criteria that you provided.	

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateGroup	The **updateGroup** command updates the common name or the description of a group.	• Parameters: **- uniqueName** Specifies the unique name value for the group for which you want to modify the properties. This parameter maps to the uniqueName property in virtual member manager. (String, required) **- cn** Specifies the new common name used for the group. This parameter maps to the cn property in virtual member manager. (String, optional) **- description** Specifies the new information about the group. This parameter maps to the description entity in a virtual member manager object. (String, optional) • Returns: Void if the command is successful. Returns an exception if the command fails.	**Batch mode example usage:** • Using Jacl: `$AdminTask updateGroup {-uniqueName cn=operators,cn=groups,dc=yourco,dc=com -cn groupA}` • Using Jython string: `AdminTask.updateGroup ('[-uniqueName cn=operators,cn=groups,dc=yourco,dc=com -cn groupA]')` • Using Jython list: `AdminTask.updateGroup (['-uniqueName', 'cn=operators,cn=groups,dc=yourco,dc=com', '-cn', 'groupA'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateGroup {-interactive}` • Using Jython string: `AdminTask.updateGroup ('[-interactive]')` • Using Jython list: `AdminTask.updateGroup (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateUser	The **updateUser** command updates the following properties: uniqueName, uid, password, cn, sn, or ibm-primaryEmail.	• Parameters: **- uniqueName** Specifies the unique name value for the user for which you want to modify the properties. This parameter maps to the uniqueName property in virtual member manager. (String, required) **- uid** Specifies the new unique ID value for the user. This parameter maps to the uid property in virtual member manager. (String, optional) **- password** Specifies the new password for the user. This parameter maps to the password property in virtual member manager. (String, optional) **- confirmPassword** Specifies the password again to validate how it was entered on the password parameter. This parameter maps to the password property in virtual member manager. (String, optional) **- cn** Specifies the new first name or given name of the user. This parameter maps to the cn property in virtual member manager. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask updateUser {-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com -uid 123}` • Using Jython string: `AdminTask.updateUser ('[-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com -uid 123]')` • Using Jython list: `AdminTask.updateUser (['-uniqueName', 'uid=dmeyers,cn=users,dc=yourco,dc=com', '-uid', '123'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask updateUser {-interactive}` • Using Jython string: `AdminTask.updateUser ('[-interactive]')` • Using Jython list: `AdminTask.updateUser (['-interactive'])`

Table 27. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- surname Specifies the new last name or family name of the user. This parameter maps to the sn property in virtual member manager. (String, optional) **- ibm-primaryEmail** Specifies the new e-mail address of the user. This parameter maps to the mail property in virtual member manager. (String, optional) • Returns: Void if the command is successful. Returns an exception if the command fails.	

Commands for the KeyStoreCommands group of the AdminTask object

Use the commands in the KeyStoreCommands group to create or delete key stores. The commands for this group do not require a target object. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the KeyStoreCommands group of the AdminTask object:

Table 28.

Command name:	Description:	Parameters and return values:	Examples:
changeKey Store Password	The **changeKey Store Password** command changes the password on the key store.	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- keyStorePassword** The password that protects the key store that you want to change. (String, required) **- newkeyStorePassword** The new password that protects the key store. (String, required) **- newkeyStorePasswordVerify** The new password that protects the key store. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask changeKeyStore Password {-keyStoreName testKS –keyStorePassword testpwd –newKeyStorePassword keyPWD –newKeyStorePasswordVerify keyPWD}` • Using Jython string: `AdminTask.changeKeyStore Password ('[-keyStoreName testKS –keyStorePassword testpwd –newKeyStorePassword keyPWD –newKeyStorePassword Verify keyPWD]')` • Using Jython list: `AdminTask.changeKeyStore Password (['-keyStoreName', 'testKS', '–keyStorePassword', 'testpwd', '–newKeyStore Password', 'keyPWD', '–new KeyStorePasswordVerify', 'keyPWD'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask changeKeyStore Password {-interactive}` • Using Jython string: `AdminTask.changeKeyStore Password ('[-interactive]')` • Using Jython list: `AdminTask.changeKeyStore Password (['-interactive'])`

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
change Multiple KeyStore Passwords	The **change Multiple KeyStore Passwords** command updates all of the key stores in the configuration that have a give password and changed them to a new password. This is useful because when you create key store files on the system, they will have `WebAS` as a password by default.	• Parameters: **- keyStorePassword** Specifies the name of the password that you want to change. (String, required) **- newKeyStorePassword** Specifies the new password that you will use to access the key store. (String, required) **- newKeyStorePasswordVerify** Confirms the new key store password. (String, required) • Returns: A list of key store aliases that where changed	**Batch mode example usage:** • Using Jacl: `$AdminTask changeMultiple KeyStorePasswords {-key StorePassword WebAS -new KeyStorePassword newpwd -newKeyStorePassword Verify newpwd}` • Using Jython string: `AdminTask.changeMultiple KeyStorePasswords ('[-key StorePassword WebAS -new KeyStorePassword newpwd -newKeyStorePasswordVerify newpwd]')` • Using Jython list: `AdminTask.changeMultiple KeyStorePasswords (['-key StorePassword', 'WebAS', '-newKeyStorePassword', 'newpwd', '-newKeyStore PasswordVerify', 'newpwd'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask changeMul tipleKeyStorePasswords {-interactive}` • Using Jython string: `AdminTask.changeMulti pleKeyStorePasswords ('[-interactive]')` • Using Jython list: `AdminTask.changeMulti pleKeyStorePasswords (['-interactive'])`

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createKeyStore	The **createKeyStore** command creates the key store settings in the configuration and the key store database.	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreType** The implementation of the key store management. (String, required) **- keyStoreLocation** The location of the key store. For file based, the location is the files system path to the key store database. For hardware key store, the location is the path to the token library. (String, required) **- keyStorePassword** The password that protects the key store. (String, required) **- keyStorePasswordVerify** The password that protects the key store. (String, required) **- keyStoreProvider** The provider used to implement the key store. (String, optional) **- isKeyStoreFileBased** Set the value of this parameter to `true` if the key store is file based. Set the value of this parameter to `false` for hardware crypto key stores. (Boolean, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createKeyStore {-keyStoreName testKS –location c:\temp\test KeyFile.p12 keyStorePass word testpwd –keyStore PasswordVerify testpwd –isKeyStoreFileBased true –keyStoreInitAtStartup true –keyStoreReadOnly false}` • Using Jython string: `AdminTask.createKeyStore ('[-keyStoreName testKS –location c:\temp\testKey File.p12 keyStorePassword testpwd –keyStorePass wordVerify testpwd –isKey StoreFileBased true –key StoreInitAtStartup true –keyStoreReadOnly false]')` • Using Jython list: `AdminTask.createKeyStore (['-keyStoreName', 'testKS', '–location', 'c:\temp\test KeyFile.p12', 'keyStorePass word', 'testpwd', '–keySto rePasswordVerify', 'testpwd', '–isKeyStoreFileBased', 'true', '–keyStoreInitAt Startup', 'true', '–key StoreReadOnly', 'false'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createKey Store {-interactive}` • Using Jython string: `AdminTask.createKeySt ore ('[-interactive]')` • Using Jython list: `AdminTask.createKeyS tore (['-interactive'])`

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		- keyStoreHostList A list of host names that indicate from where the key store is remotely managed, separated by commas. (String, optional) **- keyStoreInitAtStartup** Set the value of this parameter to `true` if the key store is initialized at startup. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- keyStoreReadOnly** Set the value of this parameter to `true` if you cannot write to the key store. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- keyStoreStashFile** Set the value of this parameter to `true` if you want to create stash files for CMS type key store. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- scopeName** The name of the scope. (String, optional) **- enableCryptoOperations** Specifies if the key store object will be used for hardware cryptographic operations or not. The default value is `false`. (Boolean, optional) • Returns: The configuration object name of the key store object that you created.	

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createCMSKeyStore	The **createCMSKeyStore** command creates a CMS key store database and the key store settings in the configuration.	• Parameters: **- cmsKeyStoreURI** The URI of the CMS key store. (String, required) **- pluginHostName** The host name of the plug-in. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask createCMSKeyStore` • Using Jython: `AdminTask.createCMSKeyStore()` **Interactive mode example usage:** • Using Jacl: `$AdminTask createCMSKeyStore {-interactive}` • Using Jython string: `AdminTask.createCMSKeyStore ('[-interactive]')` • Using Jython list: `AdminTask.createCMSKeyStore (['-interactive'])`
deleteKeyStore	The **deleteKeyStore** command deletes the settings of a key store from the configuration and the key store file.	• Parameters: **- name** The name that uniquely identifies the key store that you want to delete. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteKeyStore {-name testKS}` • Using Jython string: `AdminTask.deleteKeyStore ('[-name testKS]')` • Using Jython list: `AdminTask.deleteKeyStore (['-name', 'testKS'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteKeyStore {-interactive}` • Using Jython string: `AdminTask.deleteKeyStore ('[-interactive]')` • Using Jython list: `AdminTask.deleteKeyStore (['-interactive'])`

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
exchangeSigners	The **exchangeSigners** command exchange signer certificate between key stores.	• Parameters: **- keyStoreName1** The name that uniquely identifies a key store. You must specify a second key store name using the keyStoreName2 parameter. (String, required) **- keyStoreScope1** The scope name of the key store that you specified with the keyStoreName1 parameter. (String, required) **- certificateAlaisList1** A list of aliases separated by a comma. (String, optional) **- keyStoreName2** The name that uniquely identifies a key store. You must specify a second key store name using the keyStoreName1 parameter. (String, required) **- keyStoreScope2** The scope name of the key store that you specified with the keyStoreName2 parameter. (String, required) **- certificateAliasList2** A list of aliases separated by a comma. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask exchangeSigners {-keyStoreName1 testKS –certificateAliasList1 testCert1 –keyStoreName2 secondKS –certificate AlaisList2 certAlis}` • Using Jython string: `AdminTask.exchangeSigners ('[-keyStoreName1 testKS –certificateAliasList1 testCert1 –keyStoreName2 secondKS –certificateAlais List2 certAlis]')` • Using Jython list: `AdminTask.exchangeSigners (['-keyStoreName1', 'testKS', '–certificateAliasList1', 'testCert1', '–keyStoreName 2', 'secondKS', '–certifica teAlaisList2', 'certAlis'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask exchangeSigners {-interactive}` • Using Jython string: `AdminTask.exchangeSigners ('[-interactive]')` • Using Jython list: `AdminTask.exchangeSigners (['-interactive'])`

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getKeyStoreInfo	The **getKeyStoreInfo** command displays the settings of a particular key store.	• Parameters: **- name** The name that uniquely identifies the key store. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: The settings of the key store that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask getKeyStore {-name testKS}` • Using Jython string: `AdminTask.getKeyStore ('[-name testKS]')` • Using Jython list: `AdminTask.getKeyStore (['-name', 'testKS'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getKeyStore Info {-interactive}` • Using Jython string: `AdminTask.getKeyStore Info ('[-interactive]')` • Using Jython list: `AdminTask.getKeyStore Info (['-interactive'])`

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listKeyFileAliases	The **listKeyFileAliases** command lists the certificates in a key store file.	• Parameters: **- keyFilePath** The path of the key file. (String, required) **- keyFilePassword** The password for the key file. (String, required) **- keyFileType** The key file type. (String, required) • Returns: A list of certificate aliases.	**Batch mode example usage:** • Using Jacl: `$AdminTask listKeyFile Aliases {-keyFilePaht c:\temp\testKeyFile.p12 –keyFilePassword testPwd –keyFileType PKCS12}` • Using Jython string: `AdminTask.listKeyFile Aliases ('[-keyFilePaht c:\temp\testKeyFile.p12 –keyFilePassword testPwd –keyFileType PKCS12]')` • Using Jython list: `AdminTask.listKeyFile Aliases (['-keyFilePaht', 'c:\temp\testKeyFile.p12', '–keyFilePassword', 'test Pwd', '–keyFileType', 'PKCS12'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listKeyFileA liases {-interactive}` • Using Jython string: `AdminTask.listKeyFileA liases ('[-interactive]')` • Using Jython list: `AdminTask.listKeyFileA liases (['-interactive'])`

Table 28. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listKeyStores	The **listKeyStores** command lists the key store for a particular scope.	• Parameters: **- displayObjectName** Set the value of this parameter to `true` to list the key store configuration objects within a scope. Set the value of this parameter to `false` to list the strings that contain the key store name and management scope. (String, optional) **- scopeName** The name of the scope. (String, optional) • Returns: A list of key stores.	**Batch mode example usage:** • Using Jacl: `$AdminTask listKeyStores` • Using Jython: `AdminTask.listKeyStores()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listKeyStores {-interactive}` • Using Jython string: `AdminTask.listKeyStores ('[-interactive]')` • Using Jython list: `AdminTask.listKeyStores (['-interactive'])`
listKeyStoresTypes	The **listKeyStoresTypes** command lists all valid key store types.	• Parameters: None • Returns: A list of key store types.	**Batch mode example usage:** • Using Jacl: `$AdminTask listKeyStoreTypes` • Using Jython: `AdminTask.listKeyStoreTypes()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listKeyStores Types {-interactive}` • Using Jython string: `AdminTask.listKeyStores Types ('[-interactive]')` • Using Jython list: `AdminTask.listKeyStores Types (['-interactive'])`

Commands for the SSLConfigCommands group of the AdminTask object

Use the commands in the SSLConfigCommands group to create and delete SSL configurations. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SSLConfigCommands group of the AdminTask object:

Table 29.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createSSLConfig	The **createSSLConfig** command creates an SSL configuration that is based on key store and trust store settings. You can use the SSL configuration settings to make the SSL connections.	None	• Parameters: **- alias** The name of the alias. (String, required) **- scopeName** The name of the scope. (String, optional) **- clientKeyAlias** The certificate alias name for the client. (String, optional) **- serverKeyAlias** The certificate alias name for the server. (String, optional) **- type** The type of SSL configuration. (String, optional) **- clientAuthentication** Set the value of this parameter to `true` to request client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- securityLevel** The cipher group that you want to use. Valid values include: `HIGH`, `MEDIUM`, `LOW`, and `CUSTOM`. (String, optional) **- enabledCiphers** A list of ciphers used during SSL handshake. (String, optional) **- jsseProvider** One of the JSSE providers. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createSSLConfig {-alias testSSLCfg -clientKeyAlias key1 -serverKeyAlias key2 -trustStoreName trustKS –keyStoreName testKS -keyManagerName testKeyMgr}` • Using Jython string: `AdminTask.createSSLConfig ('[-alias testSSLCfg -clientKeyAlias key1 -serverKeyAlias key2 -trustStoreName trustKS –keyStoreName testKS -keyManagerName testKeyMgr]')` • Using Jython list: `AdminTask.createSSLConfig (['-alias', 'testSSLCfg', '-clientKeyAlias', 'key1', '-serverKeyAlias', 'key2', '-trustStoreName', 'trustKS', '–keyStoreName', 'testKS', '-keyManagerName', 'testKeyMgr'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createSSLConfig {-interactive}` • Using Jython string: `AdminTask.createSSLConfig ('[-interactive]')` • Using Jython list: `AdminTask.createSSLConfig (['-interactive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			-clientAuthenticationSupported Set the value of this parameter to `true` to support client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- sslProtocol** The protocol type for the SSL handshake. Valid values include: `SSL_TLS`, `SSL`, `SSLv2`, `SSLv3`, `TLS`, `TLSv1`. (String, optional) **-trustManagerObjectName** A list of trust managers separated by commas. (String, optional) **- trustStoreNames** The key store that holds trust information used to validate the trust from remote connections. (String, required) **-trustStoreScopeName** The management scope name of the trust store. (String, optional)	

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- keyStoreName The key store that holds the personal certificates that provide identity for the connection. (String, required) **- keyStoreScopeName** The management scope name of the key store. (String, optional) **- sslKeyRingName** Specifies a system SSL (SSSL) key ring name. The value for this parameter has no affect unless the SSL configuration type is SSSL. (String, optional) • Returns: The configuration object name of the SSL configuration object that you created.	

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createSSL Config Property	The **createSSL Config Property** command creates a property for an SSL configuration. Use this command to set SSL configuration settings that are different than the settings in the SSL configuration object.	None	• Parameters: **- sslConfigAliasName** The alias name of the SSL configuration. (String, required) **- scopeName** The name of the scope. (String, optional) **- propertyName** The name of the property. (String, required) **- propertyValue** The value of the property. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask createSSLConfig Property {-sslConfigAlias Name NodeDefaultSSLSettings -scopeName (cell):localhost Node01Cell:(node):localhost Node01 -propertyName test. property -propertyValue testValue}` • Using Jython string: `AdminTask.createSSLConfig Property ('[-sslConfig AliasName NodeDefaultSSL Settings -scopeName (cell) :localhostNode01Cell:(node) :localhostNode01 -property Name test.property -property Value testValue]')` • Using Jython list: `AdminTask.createSSLConfig Property (['-sslConfigAlias Name', 'NodeDefaultSSL Settings', '-scopeName', '(cell):localhostNode01Cell: (node):localhostNode01', '-propertyName', 'test. property', '-propertyValue', 'testValue'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createSSLConfig Property {-interactive}` • Using Jython string: `AdminTask.createSSLConfig Property ('[-interactive]')` • Using Jython list: `AdminTask.createSSLConfig Property (['-interactive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteSSLConfig	The **deleteSSLConfig** command deletes the SSL configuration object that you specify from the configuration.	None	• Parameters: **- alias** The name of the alias. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteSSLConfig {-alias NodeDefaultSSL Settings -scopeName (cell ):localhostNode01Cell:( node):localhostNode01}` • Using Jython string: `AdminTask.deleteSSLConfig ('[-alias NodeDefaultSSL Settings -scopeName (cell) :localhostNode01Cell:(no de):localhostNode01]')` • Using Jython list: `AdminTask.deleteSSLConfig (['-alias', 'NodeDefault SSLSettings', '-scopeName', '(cell):localhostNode01 Cell:(node):localhost Node01'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteSSLConfig {-interactive}` • Using Jython string: `AdminTask.deleteSSLConfig ('[-interactive]')` • Using Jython list: `AdminTask.deleteSSLConfig (['-interactive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getSSLConfig	The **getSSLConfig** command obtains information about an SSL configuration and displays the settings.	None	• Parameters: **- alias** The name of the alias. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: Information about the SSL configuration that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask getSSLConfig {-alias NodeDefaultSSL Settings -scopeName (cell):localhostNode01 Cell:(node):localhost Node01 }` • Using Jython string: `AdminTask.getSSLConfig ('[-alias NodeDefault SSLSettings -scopeName (cell):localhostNode01 Cell:(node):localhost Node01]')` • Using Jython list: `AdminTask.getSSLConfig (['-alias', 'Node DefaultSSLSettings', '-scopeName', '(cell): localhostNode01Cell: (node):localhostNode01'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getSSLConfig {-interactive}` • Using Jython string: `AdminTask.getSSLConfig ('[-interactive]')` • Using Jython list: `AdminTask.getSSLConfig (['-interactive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getSSLConfig Properties	The **getSSLConfig Properties** command obtains information about SSL configuration properties.	None	• Parameters: **- alias** The name of the alias. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: Information about SSL configuration properties.	**Batch mode example usage:** • Using Jacl: `$AdminTask getSSLConfig Properties {-sslConfig AliasName NodeDefault SSLSettings -scopeName (cell):localhostNode01 Cell:(node):localhost Node01}` • Using Jython string: `AdminTask.getSSLConfig Properties ('[-sslCon figAliasName NodeDefa ultSSLSettings -scope Name (cell):localhost Node01Cell:(node):loc alhostNode01]')` • Using Jython list: `AdminTask.getSSLConfig Properties (['-sslCon figAliasName', 'NodeDe faultSSLSettings', '-scopeName', '(cell): localhostNode01Cell: (node):localhostNode01'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getSSLConfig Properties {-interactive}` • Using Jython string: `AdminTask.getSSLConfig Properties ('[-intera ctive]')` • Using Jython list: `AdminTask.getSSLConfig Properties (['-intera ctive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLCiphers	The **listSSLCiphers** command lists the SSL ciphers.	None	• Parameters: **- sslConfigAliasName** The alias name of the SSL configuration. (String, required) **- scopeName** The name of the scope. (String, optional) **- securityLevel** The cipher group that you want to use. Valid values include: `HIGH`, `MEDIUM`, `LOW`, and `CUSTOM`. (String, required) **- provider** (String, optional) • Returns: A list of SSL ciphers.	**Batch mode example usage:** • Using Jacl: `$AdminTask listSSLCiphers {-sslConfigAliasName test SSLCfg -securityLevel HIGH}` • Using Jython string: `AdminTask.listSSLCiphers ('[-sslConfigAliasName testSSLCfg -security Level HIGH]')` • Using Jython list: `AdminTask.listSSLCiphers (['-sslConfigAliasName', 'testSSLCfg', '-security Level', 'HIGH'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listSSLCiphers {-interactive}` • Using Jython string: `AdminTask.listSSLCiphers ('[-interactive]')` • Using Jython list: `AdminTask.listSSLCiphers (['-interactive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLConfig	The **listSSLConfig** command lists the defined SSL configurations within a management scope.	None	• Parameters: **- scopeName** The name of the scope. (String, optional) **- displayObjectName** Set the value of this parameter to `true` to list the SSL configuration objects within the scope. Set the value of this parameter to `false` to list the strings that contain the SSL configuration alias and management scope. (Boolean, optional) • Returns: A list of the defined SSL configurations.	**Batch mode example usage:** • Using Jacl: `$AdminTask listSSLConfig {-scopeName (cell): localhostNode01Cell:(node): localhostNode01 -displayObjectName true}` • Using Jython string: `AdminTask.listSSLConfig ('[-scopeName (cell): localhostNode01Cell: (node):localhostNode01 -displayObjectName true]')` • Using Jython list: `AdminTask.listSSLConfig (['-scopeName', '(cell) :localhostNode01Cell: (node):localhostNode01', '-displayObjectName', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listSSLConfig {-interactive}` • Using Jython string: `AdminTask.listSSLConfig ('[-interactive]')` • Using Jython list: `AdminTask.listSSLConfig (['-interactive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLConfig Properties	The **listSSLConfig Properties** command lists the properties for a SSL configuration.	None	• Parameters: **- sslConfigAliasName** The alias name of the SSL configuration. (String, required) **- scopeName** The name of the scope. (String, optional) **- displayObjectName** Set the value of this parameter to `true` to list the SSL configuration objects within the scope. Set the value of this parameter to `false` to list the strings that contain the SSL configuration alias and management scope. (Boolean, optional) • Returns: A list of properties.	**Batch mode example usage:** • Using Jacl: `$AdminTask listSSLConfig Property {-alias No deDefaultSSLSettings -scopeName (cell):localhostNode01Cell:(node):localhostNode01 -displayObjectName true}` • Using Jython string: `AdminTask.listSSLConfig Property ('[-alias NodeDefaultSSLSettings -scopeName (cell):localhostNode01Cell:(node):localhostNode01 -displayObjectName true]')` • Using Jython list: `AdminTask.listSSLConfigProperty (['-alias', 'No', 'deDefaultSSLSettings', '-scopeName', '(cell):localhostNode01Cell:(node):localhostNode01', '-displayObjectName', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listSSLConfig Properties {-interactive}` • Using Jython string: `AdminTask.listSSLConfig Properties ('[-interactive]')` • Using Jython list: `AdminTask.listSSLConfig Properties (['-interactive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifySSLConfig	The **modifySSLConfig** command modifies the settings of an existing SSL configuration.	None	• Parameters: **- alias** The name of the alias. (String, required) **- scopeName** The name of the scope. (String, optional) **- clientKeyAlias** The certificate alias name for the client. (String, optional) **- serverKeyAlias** The certificate alias name for the server. (String, optional) **- type** The type of SSL configuration. (String, optional) **- clientAuthentication** Set the value of this parameter to `true` to request client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- securityLevel** The cipher group that you want to use. Valid values include: `HIGH`, `MEDIUM`, `LOW`, and `CUSTOM`. (String, optional) **- enabledCiphers** A list of ciphers used during SSL handshake. (String, optional) **- jsseProvider** One of the JSSE providers. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask modifySSL Config {-alias test SSLCfg -clientKeyAli as tstKey1 -serverKey Alias tstKey2 -secur ityLevel LOW}` • Using Jython string: `AdminTask.modifySSL Config ('[-alias te stSSLCfg -clientKey Alias tstKey1 -serv erKeyAlias tstKey2 -securityLevel LOW]')` • Using Jython list: `AdminTask.modifySSL Config (['-alias', 'testSSLCfg', '-cli entKeyAlias', 'tst Key1', '-serverKey Alias', 'tstKey2', '-securityLevel', 'LOW'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifySSL Config {-interactive}` • Using Jython string: `AdminTask.modifySSL Config ('[-interact ive]')` • Using Jython list: `AdminTask.modifySSL Config (['-interact ive'])`

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			-clientAuthenticationSupported Set the value of this parameter to `true` to support client authentication. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- sslProtocol** The protocol type for the SSL handshake. Valid values include: `SSL_TLS`, `SSL`, `SSLv2`, `SSLv3`, `TLS`, `TLSv1`. (String, optional) **-trustManagerObjectNames** A list of trust managers separated by commas. (String, optional) **- trustStoreName** The key store that holds trust information used to validate the trust from remote connections. (String, optional) **-trustStoreScopeName** The management scope name of the trust store. (String, optional)	

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- keyStoreName The key store that holds the personal certificates that provide identity for the connection. (String, optional) **- keyStoreScopeName** The management scope name of the key store. (String, optional) **- sslKeyRingName** Specifies a system SSL (SSSL) key ring name. The value for this parameter has no affect unless the SSL configuration type is SSSL. (String, optional) • Returns: None	

Commands for the DescriptivePropCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the DescriptivePropCommands group of the AdminTask object:

Table 30.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createDescriptiveProp	The **createDescriptiveProp** command creates key manager settings in the configuration. Use this command during SSL handshake to determine which certificate to use.	None	• Parameters: **- parentDataType** (String, required) **- parentClassName** (String, required) **- parentScopeName** (String, optional) **- name** (String, required) **- value** (String, required) **- type** (String, required) **- displayNameKey** (String, required) **- nlsRangeKey** (String, optional) **- hoverHelpKey** (String, optional) **- range** (String, optional) **- inclusive** (Boolean, optional) **- firstClass** (Boolean, optional) • Returns: The configuration object name of the key manager object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createKeyManager {-name testKM –keyManager Class com.ibm.ws.security. ltpa.LTPAKeyPairGenerator}` • Using Jython string: `AdminTask.createKeyManager ('[-name testKM –keyManag erClass com.ibm.ws.securi ty.ltpa.LTPAKeyPairGenera tor]')` • Using Jython list: `AdminTask.createKeyManager (['-name', 'testKM', '–key ManagerClass', 'com.ibm.ws .security.ltpa.LTPAKeyPair Generator'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createDescrip tiveProp {-interactive}` • Using Jython string: `AdminTask.createDescripti veProp ('[-interactive]')` • Using Jython list: `AdminTask.createDescripti veProp (['-interactive'])`
deleteDescriptiveProp	The **deleteDescriptiveProp** command deletes key manager settings from the configuration.	None	• Parameters: **- parentDataType** (String, required) **- parentClassName** (String, required) **- parentScopeName** (String, optional) **- name** (String, required) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask deleteDescrip tiveProp {-interactive}` • Using Jython string: `AdminTask.deleteDescrip tiveProp ('[-interactive]')` • Using Jython list: `AdminTask.deleteDescrip tiveProp (['-interactive'])`

Table 30. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getDescriptiveProp	The **getDescriptiveProp** command obtains information about key manager settings.	None	• Parameters: **- parentDataType** (String, required) **- parentClassName** (String, required) **- parentScopeName** (String, optional) **- name** (String, required) • Returns: Information about key manager settings.	**Interactive mode example usage:** • Using Jacl: `$AdminTask getDescriptiveProp {-interactive}` • Using Jython string: `AdminTask.getDescriptiveProp ('[-interactive]')` • Using Jython list: `AdminTask.getDescriptiveProp (['-interactive'])`
listDescriptiveProp	The **listDescriptiveProp** command lists the key managers within a particular management scope.	None	• Parameters: **- parentDataType** (String, required) **- parentClassName** (String, required) **- parentScopeName** (String, optional) **- displayObjectName** Set the value of this parameter to `true` to list the key manager objects within the scope. Set the value of this parameter to `false` to list the strings that contain the key manager name and management scope. (Boolean, optional) • Returns: A list of key managers.	**Interactive mode example usage:** • Using Jacl: `$AdminTask listDescriptiveProp {-interactive}` • Using Jython string: `AdminTask.listDescriptiveProp ('[-interactive]')` • Using Jython list: `AdminTask.listDescriptiveProp (['-interactive'])`

Table 30. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyDescriptiveProp	The **modifyDescriptiveProp** command modifies the settings of an existing key manager.	None	• Parameters: **- parentDataType** (String, required) **- parentClassName** (String, required) **- parentScopeName** (String, optional) **- name** (String, required) **- value** (String, optional) **- type** (String, optional) **- displayNameKey** (String, optional) **- nlsRangeKey** (String, optional) **- hoverHelpKey** (String, optional) **- range** (String, optional) **- inclusive** (Boolean, optional) **- firstClass** (Boolean, optional) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask modifyDescriptiveProp {-interactive}` • Using Jython string: `AdminTask.modifyDescriptiveProp ('[-interactive]')` • Using Jython list: `AdminTask.modifyDescriptiveProp (['-interactive'])`

Commands for the TrustManagerCommands group of the AdminTask object

Use the commands in the TrustManagerCommands group to create and delete a trust manager. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the TrustManagerCommands group of the AdminTask object:

Table 31.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createTrustManager	The **createTrust ManagerInfo** command creates trust manager settings in the configuration. Use this command during SSL handshake to make trust decisions about remote endpoints.	None	• Parameters: **- name** The name that uniquely identifies the trust manager. (String, required) **- scopeName** The name of the scope. (String, optional) **- provider** The provider name of the trust manager. (String, optional) **- algorithm** The algorithm name of the trust manager. (String, optional) **- trustManagerClass** Specifies a class that implements the `javax.net.sslX509TrustManager` interface. You cannot use this parameter with the provider or algorithm parameters. (String, optional) • Returns: The configuration object name of the trust manager object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createTrust Manager {-name testTM –provider IBMJSSE2 –algorithm IbmX509}` • Using Jython string: `AdminTask.createTrust Manager ('[-name testTM –provider IBMJSSE2 –algorithm IbmX509]')` • Using Jython list: `AdminTask.createTrust Manager (['-name', 'testTM', '–provider', 'IBMJSSE2', '–algorithm', 'IbmX509'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createTrust Manager {-interactive}` • Using Jython string: `AdminTask.createTrust Manager ('[-interactive]')` • Using Jython list: `AdminTask.createTrust Manager (['-interactive'])`

Table 31. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteTrustManager	The **deleteTrustManager** command deletes the trust manager settings from the configuration.	None	• Parameters: **- name** The name that uniquely identifies the trust manager. (String, optional) **- scopeName** The name of the scope. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteTrust Manager {-name testTM}` • Using Jython string: `AdminTask.deleteTrust Manager ('[-name testTM]')` • Using Jython list: `AdminTask.deleteTrust Manager (['-name', 'testTM'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteTrust Manager {-interactive}` • Using Jython string: `AdminTask.deleteTrust Manager ('[-interactive]')` • Using Jython list: `AdminTask.deleteTrust Manager (['-interactive'])`
getTrustManager	The **getTrustManager** command obtains the setting of a trust manager.	None	• Parameters: **- name** The name that uniquely identifies the trust manager. (String, optional) **- scopeName** The name of the scope. (String, optional) • Returns: The settings of the trust manager group that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask getTrustManager {-name testTM}` • Using Jython string: `AdminTask.getTrustManager ('[-name testTM]')` • Using Jython list: `AdminTask.getTrustManager (['-name', 'testTM'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getTrustMana ger {-interactive}` • Using Jython string: `AdminTask.getTrustMana ger ('[-interactive]')` • Using Jython list: `AdminTask.getTrustMana ger (['-interactive'])`

Table 31. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listTrustManagers	The **listTrustManagers** command lists the trust managers within a particular management scope.	None	• Parameters: **- scopeName** The name of the scope. (String, optional) **- displayObjectName** Set the value of this parameter to true to list the trust manager objects within a scope. Set the value of this parameter to false to list the strings that contain the trust manager name and management scope. (Boolean, optional) • Returns: A list of trust managers that are found within the management scope that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask listTrustManagers {-displayObjectName true}` • Using Jython string: `AdminTask.listTrustManagers ('[-displayObjectName true]')` • Using Jython list: `AdminTask.listTrustManagers (['-displayObjectName', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listTrustManagers {-interactive}` • Using Jython string: `AdminTask.listTrustManagers ('[-interactive]')` • Using Jython list: `AdminTask.listTrustManagers (['-interactive'])`

Table 31. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyTrustManager	The **modifyTrustManager** command changes existing trust manager settings.	None	• Parameters: **- name** The name that uniquely identifies the trust manager. (String, required) **- scopeName** The name of the scope. (String, optional) **- provider** The provider name of the trust manager. (String, optional) **- algorithm** The algorithm name of the trust manager. (String, optional) **- trustManagerClass** Specifies a class that implements the `javax.net.sslX509TrustManager` interface. You cannot use this parameter with the provider or algorithm parameters. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask modifyTrustManager {-name testTM –trustManagerClass test.trust.manager}` • Using Jython string: `AdminTask.modifyTrustManager ('[-name testTM –trustManagerClass test.trust.manager]')` • Using Jython list: `AdminTask.modifyTrustManager (['-name', 'testTM', '–trustManagerClass', 'test.trust.manager'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifyTrustManager {-interactive}` • Using Jython string: `AdminTask.modifyTrustManager ('[-interactive]')` • Using Jython list: `AdminTask.modifyTrustManager (['-interactive'])`

Commands for the keyManagerCommands group of the AdminTask object

Use the commands in the keyManagerCommands group to manage key managers. You can use these commands to create, modify, list, or obtain information about key managers. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the keyManagerCommands group of the AdminTask object:

Table 32.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeyManager	The **createKeyManager** command creates the key manager settings in the configuration. Use this command during SSL handshake to determine which certificate alias to use.	None	• Parameters: **- name** The name that uniquely identifies the key manager. (String, required) **- scopeName** The name of the scope. (String, optional) **- provider** The provider name of the key manager. (String, optional) **- algorithm** The algorithm name of the key manager. (String, optional) **- keyManagerClass** The name of the key manager implementation class. You can not use this parameter with the provider or the algorithm parameter. (String, optional) • Returns: The configuration object name of the key manager object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createKeyManager {-name testKM –keyManagerClass com.ibm.ws.security.ltpa.LTPAKeyPairGenerator}` • Using Jython string: `AdminTask.createKeyManager ('[-name testKM –keyManagerClass com.ibm.ws.security.ltpa.LTPAKeyPairGenerator]')` • Using Jython list: `AdminTask.createKeyManager (['-name', 'testKM', '–keyManagerClass', 'com.ibm.ws.security.ltpa.LTPAKeyPairGenerator'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createKeyManager {-interactive}` • Using Jython string: `AdminTask.createKeyManager ('[-interactive]')` • Using Jython list: `AdminTask.createKeyManager (['-interactive'])`

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteKeyManager	The **deleteKeyManager** command deletes the key manager settings from the configuration.	None	• Parameters: **- name** The name that uniquely identifies the key manager. (String, optional) **- scopeName** The name of the scope. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteKeyMa nager {-name testKM}` • Using Jython string: `AdminTask.deleteKeyMa nager ('[-name testKM]')` • Using Jython list: `AdminTask.deleteKeyMa nager (['-name', 'testKM'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteKeyM anager {-interactive}` • Using Jython string: `AdminTask.deleteKeyMa nager ('[-interactive]')` • Using Jython list: `AdminTask.deleteKeyM anager (['-interactive'])`
getKeyManager	The **getKeyManager** command obtains the settings of a key manager.	None	• Parameters: **- name** The name that uniquely identifies the key manager. (String, optional) **- scopeName** The name of the scope. (String, optional) • Returns: The settings of the key manager that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask getKeyMana ger {-name testKM}` • Using Jython string: `AdminTask.getKeyMana ger ('[-name testKM]')` • Using Jython list: `AdminTask.getKeyMana ger (['-name', 'testKM'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getKeyManag er {-interactive}` • Using Jython string: `AdminTask.getKeyManag er ('[-interactive]')` • Using Jython list: `AdminTask.getKeyManag er (['-interactive'])`

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeyManagers	The **listKeyManagers** command lists the key managers within a particular management scope.	None	• Parameters: **- scopeName** The name of the scope. (String, optional) **- displayObjectName** Set the value of this parameter to `true` to list the key manager objects within the scope. Set the value of this parameter to `false` to list the strings that contain the key manager name and the management scope. (Boolean, optional) • Returns: A list of key managers.	**Batch mode example usage:** • Using Jacl: `$AdminTask listKeyManagers` • Using Jython: `AdminTask.listKeyManagers()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listKeyManagers {-interactive}` • Using Jython string: `AdminTask.listKeyManagers ('[-interactive]')` • Using Jython list: `AdminTask.listKeyManagers (['-interactive'])`

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyKeyManagers	The **modifyKeyManagers** command changes existing key manager settings.	None	• Parameters: **- name** The name that uniquely identifies the key manager. (String, required) **- scopeName** The name of the scope. (String, optional) **- provider** The provider name of the key manager. (String, optional) **- algorithm** The algorithm name of the key manager. (String, optional) **- keyManagerClass** The name of the key manager implementation class. You can not use this parameter with the provider or the algorithm parameter. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask modifyKeyManager {-name testKM –provider IBMJSSE2 –algorithm IbmX509}` • Using Jython string: `AdminTask.modifyKeyManager ('[-name testKM –provider IBMJSSE2 –algorithm IbmX509]')` • Using Jython list: `AdminTask.modifyKeyManager (['-name', 'testKM', '–provider', 'IBMJSSE2', '–algorithm', 'IbmX509'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifyKeyManagers {-interactive}` • Using Jython string: `AdminTask.modifyKeyManagers ('[-interactive]')` • Using Jython list: `AdminTask.modifyKeyManagers (['-interactive'])`

Commands for the SSLConfigGroupCommands group of the AdminTask object

Use the commands in the SSLConfigGroupCommands group to create or delete a SSL configuration group. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SSLConfigGroupCommands group of the AdminTask object:

Table 33.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createSSLConfigGroup	The **createSSLConfigGroup** command creates a SSL configuration group.	None	• Parameters: **- name** The name that uniquely identifies the SSL configuration group. (String, required) **- direction** The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) **- certificateAlias** (String, required) **- scopeName** The name of the scope. (String, optional) **- sslConfigAliasName** The alias that uniquely identifies the SSL configurations in the group. (String, required) **- sslConfigScopeName** The scope that uniquely identifies the SSL configurations in the group. (String, optional) • Returns: The configuration object name of the SSL configuration group object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createSSLConfigGroup { -name testSSLCfgGrp –direction inbound –certificateAlias alias1 –sslConfigAliasName testSSLCfg}` • Using Jython string: `AdminTask.createSSLConfigGroup ['(-name testSSLCfgGrp –direction inbound –certificateAlias alias1 –sslConfigAliasName testSSLCfg)']` • Using Jython list: `AdminTask.createSSLConfigGroup [('-name', 'testSSLCfgGrp', '–direction', 'inbound', '–certificateAlias', 'alias1', '–sslConfigAliasName', 'testSSLCfg')]` **Interactive mode example usage:** • Using Jacl: `$AdminTask createSSLConfigGroup {-interactive}` • Using Jython string: `AdminTask.createSSLConfigGroup ('[-interactive]')` • Using Jython list: `AdminTask.createSSLConfigGroup (['-interactive'])`

Table 33. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteSSLConfigGroup	The **deleteSSLConfigGroup** command deletes a SSL configuration group from the configuration.	None	• Parameters: **- name** The name that uniquely identifies the SSL configuration group. (String, required) **- direction** The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteSSLConfigGroup {-name createSSLCfgGrp –direction inbound}` • Using Jython string: `AdminTask.deleteSSLConfigGroup ['(-name createSSLCfgGrp –direction inbound)']` • Using Jython list: `AdminTask.deleteSSLConfigGroup [('-name', 'createSSLCfgGrp', '–direction', 'inbound')]` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteSSLConfigGroup {-interactive}` • Using Jython string: `AdminTask.deleteSSLConfigGroup ('[-interactive]')` • Using Jython list: `AdminTask.deleteSSLConfigGroup (['-interactive'])`

Table 33. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getSSLConfigGroup	The **getSSLConfigGroup** command returns information about a SSL configuration setting.	None	• Parameters: **- name** The name that uniquely identifies the SSL configuration group. (String, required) **- direction** The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: The settings of the SSL configuration group that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask getSSLConfig Group {-name createSSL CfgGrp –direction inbound}` • Using Jython string: `AdminTask.getSSLConfig Group ['(-name create SSLCfgGrp –direction inbound)']` • Using Jython list: `AdminTask.getSSLConfig Group [('-name', 'create SSLCfgGrp', '–direction', 'inbound')]` **Interactive mode example usage:** • Using Jacl: `$AdminTask getSSLConfig Group {-interactive}` • Using Jython string: `AdminTask.getSSLConfig Group ('[-interactive]')` • Using Jython list: `AdminTask.getSSLConfig Group (['-interactive'])`

Table 33. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLConfigGroups	The **listSSLConfigGroups** command lists the SSL configuration groups within a scope and a direction.	None	• Parameters: **- direction** The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, optional) **- scopeName** The name of the scope. (String, optional) **- displayObjectName** If you set this parameter to true, the command returns a list of all of the SSL configuration group objects within the scope. If you set this parameter to false, the command returns a list of strings that contain the SSL configuration name and management scope. (Boolean, optional) • Returns: A list of SSL configuration groups.	**Batch mode example usage:** • Using Jacl: `$AdminTask –listSSLConfigGroups {-displayObjectName true}` • Using Jython string: `AdminTask.listSSLConfigGroups ['(-displayObjectName true)']` • Using Jython list: `AdminTask.listSSLConfigGroups [('-displayObjectName' 'true')]` **Interactive mode example usage:** • Using Jacl: `$AdminTask listSSLConfigGroups {-interactive}` • Using Jython string: `AdminTask.listSSLConfigGroups ('[-interactive]')` • Using Jython list: `AdminTask.listSSLConfigGroups (['-interactive'])`

Table 33. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifySSLConfigGroup	The **modifySSLConfigGroup** command modifies the setting of an existing SSL configuration group.	None	• Parameters: **- name** The name that uniquely identifies the SSL configuration group. (String, required) **- direction** The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) **- certificateAlias** (String, optional) **- scopeName** The name of the scope. (String, optional) **- sslConfigAliasName** The alias that uniquely identifies the SSL configurations in the group. (String, optional) **- sslConfigScopeName** The scope that uniquely identifies the SSL configurations in the group. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask modifySSLConfig Group {-name createSSLCfg Grp –direction inbound –certificateAlias alias2}` • Using Jython string: `AdminTask.modifySSLConfig Group ['(-name createSSL CfgGrp –direction inbound –certificateAlias alias2)']` • Using Jython list: `AdminTask.modifySSLConfig Group [('-name', 'create SSLCfgGrp', '–direction', 'inbound', '–certificate Alias', 'alias2')]` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifySSLConf igGroup {-interactive}` • Using Jython string: `AdminTask.modifySSLConf igGroup ('[-interactive]')` • Using Jython list: `AdminTask.modifySSLConf igGroup (['-interactive'])`

Commands for the DynamicSSLConfigSelections group of the AdminTask object

Use the commands in the DynamicSSLConfigSelections group to create or delete a dynamic SSL configuration selection. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the DynamicSSLConfigSelections group of the AdminTask object:

Table 34.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createDynamic SSLConfig Selection	The **createDynamic SSLConfig Selection** command creates the configuration settings for the dynamic SSL configuration selection.	None	• Parameters: **- dynSSLConfigSelectionName** The name that uniquely identifies the dynamic SSL configuration selection. (String, required) **- scopeName** The name of the scope. (String, optional) **- dynSSLConfigSelectionDescription** The description of the dynamic SSL configuration selection. (String, optional) **- dynSSLConfigSelectionInfo** The information for the dynamic SSL configuration selection. (String, required) **- sslConfigName** The name of the SSL configuration. (String, required) **- sslConfigScope** The scope of the SSL configuration. (String, optional) **- certificateAlias** The alias name to identify the certificate. (String, required) • Returns: The configuration object name of the dynamic SSL configuration selection object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createDynamicSSLConfigSelection` • Using Jython: `AdminTask.createDynamicSSLConfigSelection()` **Interactive mode example usage:** • Using Jacl: `$AdminTask createDynamicSSLConfigSelection {-interactive}` • Using Jython string: `AdminTask.createDynamicSSLConfigSelection ('[-interactive]')` • Using Jython list: `AdminTask.createDynamicSSLConfigSelection (['-interactive'])`

Table 34. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteDynamic SSLConfig Selection	The **deleteDynamic SSLConfig Selection** command deletes the dynamic SSL configuration selection from the configuration.	None	• Parameters: None • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteDynamic SSLConfigSelection` • Using Jython: `AdminTask.deleteDynamic SSLConfigSelection()` • **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteDynamic SSLConfigSelection {-interactive}` • Using Jython string: `AdminTask.deleteDynamicSSLConfigSelection ('[-interactive]')` • Using Jython list: `AdminTask.deleteDynamic SSLConfigSelection (['-interactive'])`
getDynamic SSLConfig Selection	The **getDynamic SSLConfig Selection**command obtains information about a particular dynamic SSL configuration selection.	None	• Parameters: None • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask getDynamicSSL ConfigSelection` • Using Jython: `AdminTask.getDynamicSSL ConfigSelection()` **Interactive mode example usage:** • Using Jacl: `$AdminTask getDynamicSSL ConfigSelection {-interactive}` • Using Jython string: `AdminTask.getDynamicSSL ConfigSelection ('[-interactive]')` • Using Jython list: `AdminTask.getDynamicSSL ConfigSelection (['-interactive'])`

Table 34. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listDynamic SSLConfig Selections	The **listDynamic SSLConfig Selections**command lists the configuration objects name for a dynamic SSL configuration selection.	None	• Parameters: None • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask listDynamic SSLConfigSelection` • Using Jython: `AdminTask.listDynamic SSLConfigSelection()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listDynamicSSL ConfigSelections {-intera ctive}` • Using Jython string: `AdminTask.listDynamicSSL ConfigSelections ('[-inte ractive]')` • Using Jython list: `AdminTask.listDynamicSSL ConfigSelections (['-int eractive'])`

Commands for the ManagementScopeCommands group of the AdminTask object

Use the commands in the ManagementScopeCommands group to create or delete a management scope. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the ManagementScopeCommands group of the AdminTask object:

Table 35.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createManagementScope	The **createManagementScope** command creates a management scope setting in the configuration.	None	• Parameters: **- scopeName** The name that uniquely identifies the management scope. (String, required) **- scopeType** The type of the management scope. Valid types include cell, node, nodegroup, cluster, server, and endpoint. (String, optional) • Returns: The configuration object name of the management scope object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createManagementScope {-name (cell):localhostNode01Cell –scopeType cell}` • Using Jython string: `AdminTask.createManagementScope ['(-name (cell):localhostNode01Cell –scopeType cell)']` • Using Jython list: `AdminTask.createManagementScope [('-name', '(cell):localhostNode01Cell', '–scopeType', 'cell')]` **Interactive mode example usage:** • Using Jacl: `$AdminTask createManagementScope {-interactive}` • Using Jython string: `AdminTask.createManagementScope ('[-interactive]')` • Using Jython list: `AdminTask.createManagementScope (['-interactive'])`

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteManagementScope	The **deleteManagementScope** command deletes a management object from the configuration.	None	• Parameters: **- scopeName** The name that uniquely identifies the management scope. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteManagementScope {-scopeName (cell):localhostNode01Cell}` • Using Jython string: `AdminTask.deleteManagementScope ('[-scopeName (cell):localhostNode01Cell]')` • Using Jython list: `AdminTask.deleteManagementScope (['-scopeName', '(cell):localhostNode01Cell'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteManagementScope {-interactive}` • Using Jython string: `AdminTask.deleteManagementScope ('[-interactive]')` • Using Jython list: `AdminTask.deleteManagementScope (['-interactive'])`

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getManagementScope	The **getManagementScope** command displays the setting of a management scope object.	None	• Parameters: **- scopeName** The name that uniquely identifies the management scope. (String, required) • Returns: The settings of the management scope object.	**Batch mode example usage:** • Using Jacl: `$AdminTask getManageme ntScope {-scopeName (ce ll):localhostNode01Cell}` • Using Jython string: `AdminTask.getManagement Scope ('[-scopeName (ce ll):localhostNode01Cell]')` • Using Jython list: `AdminTask.getManagement Scope (['-scopeName', '(cell):localhostNode01 Cell'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getManagement Scope {-interactive}` • Using Jython string: `AdminTask.getManagement Scope ('[-interactive]')` • Using Jython list: `AdminTask.getManagement Scope (['-interactive'])`

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listManagementScopes	The **listManagementScopes** command lists the management scopes in the configuration.	None	• Parameters: **-displayObjectName** Set the value to true to display the object names of the management scope. (Boolean, optional) • Returns: A list that contains all of the management scope names.	**Batch mode example usage:** • Using Jacl: `$AdminTask listManagem entScopes { -name testKM –provider IBMJSSE2 –alg orithm IbmX509}` • Using Jython string: `AdminTask.listManageme ntScopes ('[-name test KM –provider IBMJSSE2 –algorithm IbmX509]')` • Using Jython list: `AdminTask.listManageme ntScopes (['-name', 'testKM', '–provider', 'IBMJSSE2', '–algorithm', 'IbmX509'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listManagem entScopes {-interactive}` • Using Jython string: `AdminTask.listManagem entScopes ('[-interac tive]')` • Using Jython list: `AdminTask.listManagem entScopes (['-interac tive'])`

Commands for the WSCertExpMonitorCommands group of the AdminTask object

Use the commands in the WSCertExpMonitorCommands group to start or update the certificate expiration monitor. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the WSCertExpMonitorCommands group of the AdminTask object:

Table 36.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createWSCert ExpMonitor	The **createWSCert ExpMonitor** command creates the certificate expiration monitor settings in the configuration.	None	• Parameters: **- name** The name that uniquely identifies the certificate expiration monitor. (String, required) **- autoReplace** Set the value of this parameter to true if you want to replace a certificate within a certificate expiration date. If not, set the value of this parameter to false. (Boolean, required) **- deleteOld** Set the value of this parameter to true if you want to delete an old certificate during certificate expiration monitoring. If not, set the value of this parameter to false. (Boolean, required) **- daysBeforeNotification** The number of days before a certificate expires that you want to be notified of the expiration. (Integer, required) **- wsScheduleName** The name of the scheduler to use for certificate expiration. (String, required)	**Batch mode example usage:** • Using Jacl: `$AdminTask createWSCert ExpMonitor {-name test CertMon –autoReplace true –deleteOld true –days BeforeNotification 30 –wsScheduleName testSch edule –wsNotificationName testNotifier –isEnabled false}` • Using Jython string: `AdminTask.createWSCert ExpMonitor ('[-name tes tCertMon –autoReplace true –deleteOld true –daysBeforeNotification 30 –wsScheduleName test Schedule –wsNotificati onName testNotifier –isEnabled false]')` • Using Jython list: `AdminTask.createWSCert ExpMonitor (['-name', 'testCertMon', '–autoRe place', 'true', '–delete Old', 'true', '–daysBe foreNotification', '30', '–wsScheduleName', 'te stSchedule', '–wsNotifi cationName', 'testNotif ier', '–isEnabled', 'false'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createWSCert ExpMonitor {-interactive}` • Using Jython string: `AdminTask.createWSCert ExpMonitor ('[-interac tive]')` • Using Jython list: `AdminTask.createWSCert ExpMonitor (['-interac tive'])`

Table 36. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- wsNotificationName The name of the notifier to use for certificate expiration. (String, required) **- isEnabled** Set the value of this parameter to true if the certificate expiration monitor is enabled. If not, set the value of this parameter to false. (Boolean, optional) • Returns: The configuration object name of the certificate expiration monitor object that you created.	
deleteWSCert ExpMonitor	The **deleteWSCert ExpMonitor**command deletes the settings of a scheduler from the configuration.	None	• Parameters: **- name** The name that uniquely identifies the certificate expiration monitor. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteWSCert ExpMonitor {-name test CertMon}` • Using Jython string: `AdminTask.deleteWSCertE xpMonitor ('[-name test CertMon]')` • Using Jython list: `AdminTask.deleteWSCert ExpMonitor (['-name', 'testCertMon'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteWSCert ExpMonitor {-interactive}` • Using Jython string: `AdminTask.deleteWSCert ExpMonitor ('[-interac tive]')` • Using Jython list: `AdminTask.deleteWSCert ExpMonitor (['-interac tive'])`

Table 36. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getWSCertExpMonitor	The **getWSCertExpMonitor** command displays the settings of a particular scheduler.	None	• Parameters: **- name** The name that uniquely identifies the certificate expiration monitor. (String, required) • Returns: The settings of the scheduler that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask getWSCertExp Monitor {-name testCertMon}` • Using Jython string: `AdminTask getWSCertExp Monitor ('[-name testCe rtMon]')` • Using Jython list: `AdminTask getWSCertExpM onitor (['-name', 'test CertMon'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getWSCertExp Monitor {-interactive}` • Using Jython string: `AdminTask.getWSCertExp Monitor ('[-interactive]')` • Using Jython list: `AdminTask.getWSCertExp Monitor (['-interactive'])`

Table 36. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listWSCertExpMonitor	The **listWSCertExpMonitor** command lists the scheduler in the configuration.	None	• Parameters: **-displayObjectNames** If you set the value of this parameter to true, the command returns the certificate expiration monitor configuration object. If you set the value of this parameter to false, the command returns the name of the certificate expiration monitor. (Boolean, optional) • Returns: The scheduler in the configuration.	**Batch mode example usage:** • Using Jacl: `$AdminTask listWSCertExp Monitor {-displayObject Name false}` • Using Jython string: `AdminTask.listWSCertExp Monitor ('[-displayObject Name false]')` • Using Jython list: `AdminTask.listWSCertExp Monitor (['-displayObject Name', 'false'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listWSCert ExpMonitor {-interactive}` • Using Jython string: `AdminTask.listWSCertExp Monitor ('[-interactive]')` • Using Jython list: `AdminTask.listWSCertExp Monitor (['-interactive'])`

Table 36. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyWSCert ExpMonitor	The **modifyWSCert ExpMonitor**command changes the setting of an existing scheduler.	None	• Parameters: **- name** The name that uniquely identifies the certificate expiration monitor. (String, required) **- autoReplace** Set the value of this parameter to true if you want to replace a certificate within a certificate expiration date. If not, set the value of this parameter to false. (Boolean, required) **- deleteOld** Set the value of this parameter to true if you want to delete an old certificate during certificate expiration monitoring. If not, set the value of this parameter to false. (Boolean, required) **- daysBeforeNotification** The number of days before a certificate expires that you want to be notified of the expiration. (Integer, required) **- wsScheduleName** The name of the scheduler to use for certificate expiration. (String, required)	**Batch mode example usage:** • Using Jacl: `$AdminTask modifyWSCert ExpMonitor {-name testC ertMon –autoReplace fal se –deleteOld false –d aysBeforeNotification 20 –isEnabled true}` • Using Jython string: `AdminTask.modifyWSCert ExpMonitor ('[-name t estCertMon –autoReplac e false –deleteOld fal se –daysBeforeNotifica tion 20 –isEnabled true]')` • Using Jython list: `AdminTask.modifyWSCert ExpMonitor (['-name', 'testCertMon', '–autoR eplace', 'false', '–de leteOld', 'false', '–d aysBeforeNotification', '20', '–isEnabled', 'tr ue'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifyWSCert ExpMonitor {-interactive}` • Using Jython string: `AdminTask.modifyWSCert ExpMonitor ('[-interac tive]')` • Using Jython list: `AdminTask.modifyWSCert ExpMonitor (['-intera ctive'])`

Table 36. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- wsNotificationName The name of the notifier to use for certificate expiration. (String, required) **- isEnabled** Set the value of this parameter to true if the certificate expiration monitor is enabled. If not, set the value of this parameter to false. (Boolean, optional) • Returns: None	
startCertificate ExpMonitor	The **startCertificate ExpMonitor**command performs certificate monitoring. This command visits all key stores and checks to see if they are within certificate expiration range.	None	• Parameters: None • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask startCertificateExpMonitor` • Using Jython: `AdminTask.startCertificateExpMonitor()` **Interactive mode example usage:** • Using Jacl: `$AdminTask startCertificateExpMonitor {-interactive}` • Using Jython string: `AdminTask.startCertificateExpMonitor ('[-interactive]')` • Using Jython list: `AdminTask.startCertificateExpMonitor (['-interactive'])`

Commands for the KeySetGroupCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the KeySetGroupCommands group of the AdminTask object:

Table 37.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeySetGroup	The **createKeySetGroup** command creates the key set group settings in the configuration. Use this command to manage groups of public, private, and shared keys.	None	• Parameters: **- name** The name that uniquely identifies the key set group. (String, required) **- scopeName** The name of the scope. (String, optional) **- autoGenerate** Set the value of this parameter to true if you want to automatically generate keys. If not, set the value to false. (Boolean, optional) **- wsScheduleName** The name of the scheduler to use to perform key generation. (String, required) **- keySetObjectNames** A list of key set configuration names separated by colons (:). (String, required) • Returns: The configuration object name of the key set group object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createKeySet Group {-name keySetGrp –autoGenerate true –wsS cheduleName testSchedule –keySetObjectNames test KeySet(cells/localhost Node01Cell\|security.xml# KeySet_1130354347825)}` • Using Jython string: `AdminTask.createKeySetGr oup('[-name keySetGrp – autoGenerate true –wsSch eduleName testSchedule –keySetObjectNames test KeySet(cells/localhostN ode01Cell\|security.xml# KeySet_1130354347825)]')` • Using Jython list: `AdminTask.createKeySetGr oup(['-name', 'keySetGrp', '–autoGenerate', 'true', '–wsScheduleName', 'test Schedule', '–keySetObject Names', 'testKeySet(cells /localhostNode01Cell\|sec urity.xml#KeySet_1130354 347825)'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createKey SetGroup {-interactive}` • Using Jython string: `AdminTask.createKeySet Group ('[-interactive]')` • Using Jython list: `AdminTask.createKeySet Group (['-interactive'])`

Table 37. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteKeySetGroup	The **deleteKeySetGroup** command deletes the settings of a key set group from the configuration.	None	• Parameters: **- name** The name that uniquely identifies the key set group. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteKeySet Group {-name keySetGrp }` • Using Jython string: `AdminTask.deleteKeySet Group ('[-name keySetGrp]')` • Using Jython list: `AdminTask.deleteKeySet Group (['-name', 'key SetGrp'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteKey SetGroup {-interactive}` • Using Jython string: `AdminTask.deleteKeySet Group ('[-interactive]')` • Using Jython list: `AdminTask.deleteKeySet Group (['-interactive'])`

Table 37. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
generateKeys ForKey SetGroup	The **generateKeys ForKey SetGroup**command generates keys for all of the keys in the key sets that make up the key set group.	None	• Parameters: **-keySetGroupName** The name of the key set group. (String, required) **-keySetGroupScope** The scope of the key set group. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask generateKey ForKeySetGroup {-keySet GroupName keySetGrp}` • Using Jython string: `AdminTask.generateKey ForKeySetGroup ('[-key SetGroupName keySetGrp]')` • Using Jython list: `AdminTask.generateKey ForKeySetGroup (['-key SetGroupName', 'keySetGrp'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask generateKeys ForKeySetGroup {-intera ctive}` • Using Jython string: `AdminTask.generateKeys ForKeySetGroup ('[-int eractive]')` • Using Jython list: `AdminTask.generateKeys ForKeySetGroup (['-inte ractive'])`

Table 37. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getKeySetGroups	The **getKeySetGroups** command displays the settings of a particular key set group.	None	• Parameters: **- name** The name that uniquely identifies the key set group. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: The settings of the specified key set group.	**Batch mode example usage:** • Using Jacl: `$AdminTask getKeySetGroup { -name keySetGrp }` • Using Jython string: `AdminTask.getKeySetGroup ('[-name keySetGrp]')` • Using Jython list: `AdminTask.getKeySetGroup (['-name', 'keySetGrp'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getKeySetGroups {-interactive}` • Using Jython string: `AdminTask.getKeySetGroups ('[-interactive]')` • Using Jython list: `AdminTask.getKeySetGroups (['-interactive'])`

Table 37. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeySetGroups	The **listKeySetGroups** command lists the key set groups for a particular scope.	None	• Parameters: **- scopeName** The name of the scope. (String, optional) **- displayObjectNames** If you set the value of this parameter to true, the command returns a list of all of the key set group objects within a scope. If you set the value of this parameter to false, the command returns a list of strings that contain the key set group name and management scope. (Boolean, optional) • Returns: A list of key set groups.	**Batch mode example usage:** • Using Jacl: `$AdminTask listKeySetGroup {-displayObjectName true}` • Using Jython string: `AdminTask.listKeySetGroup ('[-displayObjectName true]')` • Using Jython list: `AdminTask.listKeySetGroup (['-displayObjectName', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listKeySetGroups {-interactive}` • Using Jython string: `AdminTask.listKeySetGroups ('[-interactive]')` • Using Jython list: `AdminTask.listKeySetGroups (['-interactive'])`

Table 37. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyKeySetGroup	The **modifyKeySetGroup** command changes the settings of an existing key set group.	None	• Parameters: **- name** The name that uniquely identifies the key set group. (String, required) **- scopeName** The name of the scope. (String, optional) **- autoGenerate** Set the value of this parameter to true if you want to automatically generate keys. If not, set the value to false. (Boolean, optional) **- wsScheduleName** The name of the scheduler to use to perform key generation. (String, optional) **- keySetObjectNames** A list of key set configuration names separated by colons (:). (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask modifyKeySet Group {-name keySetGrp –autoGenetate false}` • Using Jython string: `AdminTask.modifyKeySet Group ('[-name keySetGrp –autoGenetate false]')` • Using Jython list: `AdminTask.modifyKeySet Group (['-name', 'key SetGrp', '–autoGenetate', 'false'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifyKeySet Group {-interactive}` • Using Jython string: `AdminTask.modifyKeySet Group ('[-interactive]')` • Using Jython list: `AdminTask.modifyKeySet Group (['-interactive'])`

Commands for the KeySetCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the KeySetCommands group of the AdminTask object:

Table 38.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeySet	The **createKeySet** command creates the key set settings in the configuration. Use this command to control key instances that have the same type.	None	• Parameters: **- name** The name that uniquely identifies the key set. (String, required) **- scopeName** The name of the scope. (String, optional) **- aliasPrefix** The prefix for the key alias when a new key generates. (String, required) **- password** The password that protects the key in the key store. (String, required) **- maxKeyReferences** The maximum number of key references returned keys from this key set. (Integer, required) **- deleteOldKeys** Set the value of this parameter to `true` to delete old keys when new keys are generated. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- keyGenerationClass** The class that is used to generate new keys in the key set. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createKeySet {-name testKeySet –alias Prefix test –password pwd –maxKeyReferences 2 –deleteOldKeys true –keyStoreName testKeyStore –isKeyPair false}` • Using Jython string: `AdminTask.createKeySet('[-name testKeySet –alias Prefix test –password pwd –maxKeyReferences 2 –deleteOldKeys true –keyStoreName testKeyStore –isKeyPair false]')` • Using Jython list: `AdminTask.createKeySet(['-name', 'testKeySet', '–aliasPrefix', 'test', '–password', 'pwd', '–maxKeyReferences', '2', '–deleteOldKeys', 'true', '–keyStoreName', 'testKeyStore', '–isKeyPair', 'false'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createKeySet {-interactive}` • Using Jython string: `AdminTask.createKeySet('[-interactive]')` • Using Jython list: `AdminTask.createKeySet(['-interactive'])`

Table 38. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- keyStoreName The key store that contains the keys. (String, required) **- keyStoreScopeName** The management scope where the key store is located. (String, optional) **- isKeyPair** Set the value of this parameter to `true` if the keys in the key set are key pairs. Otherwise, set the value of this parameter to `false`. (Boolean, optional) • Returns: The configuration object name of the key set object that you created.	
deleteKeySet	The **deleteKeySet** command deletes the settings of a key set from the configuration.	None	• Parameters: **- name** The name that uniquely identifies the key set. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteKeySet {-name testKeySet}` • Using Jython string: `AdminTask.deleteKeySet ('[-name testKeySet]')` • Using Jython list: `AdminTask.deleteKeySet (['-name', 'testKeySet'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteKeySet {-interactive}` • Using Jython string: `AdminTask.deleteKeySet ('[-interactive]')` • Using Jython list: `AdminTask.deleteKeySet (['-interactive'])`

Table 38. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
generateKey ForKeySet	The **generateKey ForKeySet** command generates keys for the keys in the key set.	None	• Parameters: **- keySetName** The name of the key set. (String, required) **- keySetScope** The scope of the key set. (String, optional) **- keySetSaveConfig** Set the value of this parameter to `true` to save the configuration of the key set. Otherwise, set the value of this parameter to `false`. (Boolean, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask generateKey ForKeySet{ -keySetName testKeySet }` • Using Jython string: `AdminTask.generateKeyFor KeySet('[-keySetName testKeySet]')` • Using Jython list: `AdminTask.generateKeyFor KeySet(['-keySetName', 'testKeySet'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask generateKeyFor KeySet {-interactive}` • Using Jython string: `AdminTask.generateKeyFo rKeySet ('[-interactive]')` • Using Jython list: `AdminTask.generateKeyFo rKeySet (['-interactive'])`
getKeySet	The **getKeySet** command displays the settings of a particular key set.	None	• Parameters: **- name** The name that uniquely identifies the key set. (String, required) **- scopeName** The name of the scope. (String, optional) • Returns: The settings of the specified key set group.	**Batch mode example usage:** • Using Jacl: `$AdminTask getKeySet {-name testKeySet}` • Using Jython string: `AdminTask.getKeySet ('[-name testKeySet]')` • Using Jython list: `AdminTask.getKeySet (['-name', 'testKeySet'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getKeySet {-interactive}` • Using Jython string: `AdminTask.getKeySet ('[-interactive]')` • Using Jython list: `AdminTask.getKeySet (['-interactive'])`

Table 38. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeySets	The **listKeySets** command lists the key sets in a particular scope.	None	• Parameters: **- scopeName** The name of the scope. (String, optional) **- displayObjectNames** Set the value of this parameter to `true` to list the key set configuration objects within the scope. Set the value of this parameter to `false` if you want to list the strings that contain the key set group name and management scope. (Boolean, optional) • Returns: The key sets for the scope that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask listKeySets {-displayObjectName true}` • Using Jython string: `AdminTask.listKeySets ('[-displayObjectName true]')` • Using Jython list: `AdminTask.listKeySets (['-displayObjectName', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listKeySets {-interactive}` • Using Jython string: `AdminTask.listKeySets ('[-interactive]')` • Using Jython list: `AdminTask.listKeySets (['-interactive'])`

Table 38. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyKeySet	The **modifyKeySet** command changes the settings of an existing key set.	None	• Parameters: **- name** The name that uniquely identifies the key set. (String, required) **- scopeName** The name of the scope. (String, optional) **- aliasPrefix** The prefix for the key alias when a new key generates. (String, optional) **- password** The password that protects the key in the key store. (String, optional) **- maxKeyReferences** The maximum number of key references returned keys from this key set. (Integer, optional) **- deleteOldKeys** Set the value of this parameter to `true` to delete old keys when new keys are generated. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- keyGenerationClass** The class that is used to generate new keys in the key set. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask modifyKeySet {-name testKeySet -maxKeyReferences 3 –deleteOldKeys false}` • Using Jython string: `AdminTask.modifyKeySet ('[-name testKeySet -maxKeyReferences 3 –deleteOldKeys false]')` • Using Jython list: `AdminTask.modifyKeySet (['-name', 'testKeySet', '-maxKeyReferences', '3', '–deleteOldKeys', 'false'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifyKeySet {-interactive}` • Using Jython string: `AdminTask.modifyKeySet ('[-interactive]')` • Using Jython list: `AdminTask.modifyKeySet (['-interactive'])`

Table 38. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- keyStoreName The key store that contains the keys. (String, optional) **- keyStoreScopeName** The management scope where the key store is located. (String, optional) **- isKeyPair** Set the value of this parameter to `true` if the keys in the key set are key pairs. Otherwise, set the value of this parameter to `false`. (Boolean, optional) • Returns: None	

Commands for the KeyReferenceCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the KeyReferenceCommands group of the AdminTask object:

Table 39.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeyReference	The **createKeyReference** command creates the key reference setting in the configuration for key set objects.	None	• Parameters: **- keySetName** The name that uniquely identifies the key set to which the key reference belongs. (String, required) **- keySetScope** The management scope of the key set. (String, optional) **- keyAlias** The alias name that identifies the key for the key set that you specify. (String, required) **- keyPassword** The password used for encrypting the key. (String, optional) **- keyPasswordVerify** The password used for encrypting the key. (String, optional) **- version** The version of the key reference. (String, optional) **- keyReferenceSaveConfig** Set the value of this parameter to `true` to save the key reference to the configuration. Otherwise, set the value to `false`. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createKeyReference {-keySetName testKeySet -keyAlias testKey –password testPWD –passwordVerify testPWD –keyReferenceSaveConfig true}` • Using Jython string: `AdminTask.createKeyReference ('[-keySetName testKeySet -keyAlias testKey –password testPWD –passwordVerify testPWD –keyReferenceSaveConfig true]')` • Using Jython list: `AdminTask.createKeyReference (['-keySetName', 'testKeySet', '-keyAlias', 'testKey', '–password', 'testPWD', '–passwordVerify', 'testPWD', '–keyReferenceSaveConfig', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createKeyReference {-interactive}` • Using Jython string: `AdminTask.createKeyReference ('[-interactive]')` • Using Jython list: `AdminTask.createKeyReference (['-interactive'])`

Table 39. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			• Returns: The configuration object name of the key reference scope object that you created.	
deleteKeyReference	The **deleteKeyReference** command deletes a key reference object from the key set object in the configuration.	None	• Parameters: **- keySetName** The name that uniquely identifies the key set to which the key reference belongs. (String, required) **- keySetScope** The management scope of the key set. (String, optional) **- keyAlias** The alias name that identifies the key for the key set that you specify. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteKeyReference { -keySetName testKeySet –keyAlias testKey }` • Using Jython string: `AdminTask.deleteKeyReference ('[-keySetName testKeySet –keyAlias testKey]')` • Using Jython list: `AdminTask.deleteKeyReference (['-keySetName', 'testKeySet', '–keyAlias', 'testKey'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteKeyReference {-interactive}` • Using Jython string: `AdminTask.deleteKeyReference ('[-interactive]')` • Using Jython list: `AdminTask.deleteKeyReference (['-interactive'])`

Table 39. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getKeyReference	The **getKeyReference** command displays the setting of a key reference object.	None	• Parameters: **- keySetName** The name that uniquely identifies the key set to which the key reference belongs. (String, required) **- keySetScope** The management scope of the key set. (String, optional) **- keyAlias** The alias name that identifies the key for the key set that you specify. (String, required) • Returns: The settings of the key reference object.	**Batch mode example usage:** • Using Jacl: `$AdminTask getKeyReference { -keySetName testKeySet –keyAlias testKey }` • Using Jython string: `AdminTask.getKeyReference ('[-keySetName testKeySet –keyAlias testKey]')` • Using Jython list: `AdminTask.getKeyReference (['-keySetName', 'testKeySet', '–keyAlias', 'testKey'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask getKeyReference {-interactive}` • Using Jython string: `AdminTask.getKeyReference ('[-interactive]')` • Using Jython list: `AdminTask.getKeyReference (['-interactive'])`

Table 39. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeyReferences	The **listKeyReferences** command lists the key references for a particular key set in the configuration.	None	• Parameters: **- keySetName** The name that uniquely identifies the key set to which the key reference belongs. (String, required) **- keySetScope** The management scope of the key set. (String, optional) • Returns: The configuration object name of the key reference scope object that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask listKeyRef ereces { -keySetName testKeySet}` • Using Jython string: `AdminTask.listKeyRefere ces ('[-keySetName test KeySet]')` • Using Jython list: `AdminTask.listKeyRefere ces (['-keySetName', 'testKeySet'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listKeyRef erences {-interactive}` • Using Jython string: `AdminTask.listKeyRefer ences ('[-interactive]')` • Using Jython list: `AdminTask.listKeyRefer ences (['-interactive'])`

Commands for the securityEnablement group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the securityEnablement group of the AdminTask object:

Table 40.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
GlobalSettings		None	• Parameters: **- secureApps** Enables application security. The default value is `true`. (Boolean, required) **- secureLocalResources** Enables Java 2 Security. The default value is `true`. (Boolean, required) **- ltpaPassword** Defines the LTPA password. Valid values include the password used by LTPA. (String, required) **- userRegistryType** Defines the type of user registry to use. Valid values include: `Embedded`, `LDAP`, `LocalOS`, or `Custom`. (String, required) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask GlobalSettings {-interactive}` • Using Jython string: `AdminTask.GlobalSettings ('[-interactive]')` • Using Jython list: `AdminTask.GlobalSettings (['-interactive'])`

Table 40. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
CustomRegistry	The **CustomRegistry** command checks that the specified name for the primary administrator does not already exist in the custom registry.	None	• Parameters: **- primaryAdminName** Represents a name of the primary administrator. The value must be a valid administrator name. (String, required) **- serverUserName** Represents the user name used to connect to internal security routines. The value must be a valid user name. (String, required) **- customRegistryClass** Represents the class name of the custom user registry implementation of the UserRegistry. The value must be a valid class name for a custom registry. (String, required) **- customProperties** Custom user registry properties. An array or name-value properties. (String, required) **- customName** The custom user registry property name. (String, required) **- customValue** The custom user registry property value that is associated with the property name. (String, required) • Returns: A value of `true` if the primary administrator does not already exist in the custom registry. Otherwise, returns a value of `false`.	**Interactive mode example usage:** • Using Jacl: `$AdminTask CustomRegistry {-interactive}` • Using Jython string: `AdminTask.CustomRegistry ('[-interactive]')` • Using Jython list: `AdminTask.CustomRegistry (['-interactive'])`

Table 40. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
EmbeddedRegistry	The **EmbeddedRegistry** command checks that the specified name for the primary administrator does not already exist in the WIM file-based registry. Returns true if it does not; false, otherwise.	None	• Parameters: **- primaryAdminName** This string represents a name of the primary administrator. The value must be a valid administrator name. (String, required) **- adminPassword** The password associated with the primary administrator. The value must be a valid administrator password. (String, required) **- serverUserName** This string represents the user name used to connect to internal security routines. The value must be a valid user name. (String, required) • Returns: A value of `true` if the specified name for the primary administrator does not already exist in the WIM file-based registry. Otherwise, returns a value of `false`.	**Interactive mode example usage:** • Using Jacl: `$AdminTask EmbeddedRegistry {-interactive}` • Using Jython string: `AdminTask.EmbeddedRegistry ('[-interactive]')` • Using Jython list: `AdminTask.EmbeddedRegistry (['-interactive'])`

Table 40. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
LDAPRegistry	The **LDAPRegistry** command checks, after all the inputs for the LDAP registry have been defined, that a connection can be made successfully to the LDAP server. An SSL connection test to LDAP is also supported using this command.	None	• Parameters: **- ldapServerType** The type of LDAP user registry to be connected to WebSphere Application Server. Valid values include: `IBM Tivoli Directory Server`, `SecureWay`, `Sun ONE`, `Domino`, `Active Directory`, `eDirectory`, `Custom`. (String, required) **- ldapHostname** The host name for the LDAP server. (String, required) **- ldapPort** The port to connect to the LDAP server. (Integer, required) **- ldapBaseDN** The base distinguished name of the directory service, the starting point for searches. (String, required) **- ldapBindDN** The bind distinguished name, used to bind to the directory server. (String, required)	**Interactive mode example usage:** • Using Jacl: `$AdminTask LDAPRegistry {-interactive}` • Using Jython string: `AdminTask.LDAPRegistry ('[-interactive]')` • Using Jython list: `AdminTask.LDAPRegistry (['-interactive'])`

Table 40. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- ldapBindPassword The password for the application server, used to bind to the directory service (String, required) **- ldapServerUserName** The user ID that is used to run WebSphere Application Server for security purposes. (String, required) • Returns: A value of `true` if the connection was successful. Otherwise, returns a value of `false`.	
LocalOSRegistry	The **LocalOSRegistry** command checks that the specified name for the primary administrator does not already exist in the LocalOS registry.	None	• Parameters: **- primaryAdminName** This string represents a name of the primary administrator. The value must be a valid administrator name. (String, required) **- serverUserName** This string represents the user name used to connect to internal security routines. The value must be a valid user name. (String, required) • Returns: A value of `true` if the specified name for the primary administrator does not already exist in the LocalOS registry. Otherwise, returns a value of `false`.	**Interactive mode example usage:** • Using Jacl: `$AdminTask LocalOSRegistry {-interactive}` • Using Jython string: `AdminTask.LocalOSRegistry ('[-interactive]')` • Using Jython list: `AdminTask.LocalOSRegistry (['-interactive'])`

Table 40. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
CurrentSettings	The **CurrentSettings** command returns a string that represents all of the existing settings set by the wizard that will be read from the workspace instead of from the configuration repository of the `security.xml` file.	None	• Parameters: **-currentWizardSettings** A properties object that contains all the current settings selected through the wizard. (Properties, required) • Returns: A string of security settings as name-value pairs.	**Interactive mode example usage:** • Using Jacl: `$AdminTask CurrentSettings {-interactive}` • Using Jython string: `AdminTask.CurrentSettings ('[-interactive]')` • Using Jython list: `AdminTask.CurrentSettings (['-interactive'])`

Commands for the CertificateRequestCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the CertificateRequestCommands group of the AdminTask object:

Table 41.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createCertificate Request	The **createCertificate Request** command creates a certificate request that is associated with a particular key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) **- certificateVersion** The certificate version. (String, required) **- certificateSize** (Integer, required) **- certificateCommonName** (String, required) **- certificateOrganization** (String, optional) **- certificateOrganizationUnit** (String, optional) **- certificateLocality** (String, optional) **- certificateState** The state code for the certificate. (String, optional) **- certificateZip** The zip code for the certificate. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createCertificateRequest {-keyStoreName testKeyStore -certificateAlias certReq -certificateSize 1024 -certificate CommonName localhost -certificate Organization testing -certificate RequestFilePath c:\temp\testCertReq.arm}` • Using Jython string: `AdminTask.createCertificateRequest ('[-keyStoreName testKeyStore -certificateAlias certReq -certificateSize 1024 -certificate CommonName localhost -certificate Organization testing -certificate RequestFilePath c:\temp\testCertReq.arm]` • Using Jython list: `AdminTask.createCertificateRequest (['-keyStoreName', 'testKeyStore', '-certificateAlias', 'certReq', '-certificateSize', '1024', '-certificateCommonName', 'localhost', '-certificateOrganization', 'testing', '-certificateRequestFilePath', 'c:\temp\testCert Req.arm'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createCertific ateRequest {-interactive}` • Using Jython string: `AdminTask.createCertific ateRequest ('[-interactive]')` • Using Jython list: `AdminTask.createCertifica teRequest (['-interactive'])`

Table 41. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- certificateCountry The country for the certificate. (String, optional) **- certificateValidDays** The amount of time in days for which the certificate is valid. (Integer, optional) **- certificateRequestFilePath** The file location of the certificate request that can be sent to a certificate authority. (String, required) • Returns: The configuration object name of the key store object that you created.	
deleteCertificate Request	The **deleteCertificate Request** command deletes a certificate request from a key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask deleteCertificateRequest {-interactive}` • Using Jython string: `AdminTask.deleteCertificateRequest ('[-interactive]')` • Using Jython list: `AdminTask.deleteCertificateRequest (['-interactive'])`

Table 41. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
extractCertificate Request	The **extractCertificate Request**command extracts a certificate request to a file.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) **- certificateRequestFilePath** The file location of the certificate request that can be sent to a certificate authority. (String, required) • Returns: A certificate request file is created that contains the extracted certificate.	**Interactive mode example usage:** • Using Jacl: `$AdminTask extractCertificateRequest {-interactive}` • Using Jython string: `AdminTask.extractCertificateRequest ('[-interactive]')` • Using Jython list: `AdminTask.extractCertificateRequest (['-interactive'])`

Table 41. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getCertificateRequest	The **getCertificateRequest** command obtains information about a particular certificate request in a key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: Information about the certificate request.	**Interactive mode example usage:** • Using Jacl: `$AdminTask getCertificate Request {-interactive}` • Using Jython string: `AdminTask.getCertificate Request ('[-interactive]')` • Using Jython list: `AdminTask.getCertificate Request (['-interactive'])`
listCertificateRequest	The **listCertificateRequest** command lists all the certificate requests associated with a particular key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) • Returns: An attribute list for each certificate request in a key store.	**Interactive mode example usage:** • Using Jacl: `$AdminTask listCertifica teRequest {-interactive}` • Using Jython string: `AdminTask.listCertifica teRequest ('[-interactive]')` • Using Jython list: `AdminTask.listCertificat eRequest (['-interactive'])`

Commands for the SignerCertificateCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SignerCertificateCommands group of the AdminTask object:

Table 42.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
addSignerCertificate	The **addSignerCertificate** command adds a signer certificate from a certificate file to a key store.	None	• Parameters: **- keyStoreName** The name of the key store where the signer certificate is located. (String, required) **- keyStoreScope** The management scope of the key store. (String, optional) **- certificateFilePath** The full path name of the file that contains the signer certificate. (String, required) **- certificateAlias** The alias name of the signer certificate in the key store. (String, required) **- base64Encoded** Set the value of this parameter to `true` if the certificate is ascii base 64 encoded. Set the value of this parameter to `false` if the certificate is binary. (String, required) • Returns: The configuration object name of the key store object that you created.	**Interactive mode example usage:** • Using Jacl: `$AdminTask addSignerCert ificate {-interactive}` • Using Jython string: `AdminTask.addSignerCert ificate ('[-interactive]')` • Using Jython list: `AdminTask.addSignerCert ificate (['-interactive'])`

Table 42. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteSignerCertificate	The **deleteSignerCertificate** command deletes a signer certificate from a key store.	None	• Parameters: **- keyStoreName** The name of the key store where the signer certificate is located. (String, required) **- keyStoreScope** The management scope of the key store. (String, optional) **- certificateAlias** The alias name of the signer certificate in the key store. (String, required) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask deleteSignerCertificate {-interactive}` • Using Jython string: `AdminTask.deleteSignerCertificate ('[-interactive]')` • Using Jython list: `AdminTask.deleteSignerCertificate (['-interactive'])`

Table 42. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
extractSignerCertificate	The **extractSignerCertificate** command extracts a signer certificate from a key store to a file.	None	• Parameters: **- keyStoreName** The name of the key store where the signer certificate is located. (String, required) **- keyStoreScope** The management scope of the key store. (String, optional) **- certificateAlias** The alias name of the signer certificate in the key store. (String, required) **- certificateFilePath** The full path name of the file that contains the signer certificate. (String, required) **- base64Encoded** Set the value of this parameter to `true` if the certificate is ascii base 64 encoded. Set the value of this parameter to `false` if the certificate is binary. (String, required) • Returns: The certificate file is created and contains the signer certificate.	**Interactive mode example usage:** • Using Jacl: `$AdminTask extractSigner Certificate {-interactive}` • Using Jython string: `AdminTask.extractSignerC ertificate ('[-interactive]')` • Using Jython list: `AdminTask.extractSignerCe rtificate (['-interactive'])`

Table 42. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getSignerCertificate	The **getSignerCertificate** command obtains information about a signer certificate from a key store.	None	• Parameters: **- keyStoreName** The name of the key store where the signer certificate is located. (String, required) **- keyStoreScope** The management scope of the key store. (String, optional) **- certificateAlias** The alias name of the signer certificate in the key store. (String, required) • Returns: Information about a signer certificate.	**Interactive mode example usage:** • Using Jacl: `$AdminTask getSignerCert ificate {-interactive}` • Using Jython string: `AdminTask.getSignerCert ificate ('[-interactive]')` • Using Jython list: `AdminTask.getSignerCert ificate (['-interactive'])`
listSignerCertificates	The **listSignerCertificates** command lists all signer certificates in a particular key store.	None	• Parameters: **- keyStoreName** The name of the key store where the signer certificate is located. (String, required) **- keyStoreScope** The management scope of the key store. (String, optional) • Returns: A list of signer certificate aliases.	**Interactive mode example usage:** • Using Jacl: `$AdminTask listSignerCer tificates {-interactive}` • Using Jython string: `AdminTask.listSignerCer tificates ('[-interactive]')` • Using Jython list: `AdminTask.listSignerCer tificates (['-interactive'])`

Table 42. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
retrieveSignerFromPort	The **retrieveSignerFromPort** command retrieves a signer from a remote host and stores the signer in a key store.	None	• Parameters: **- host** The host name of the system from where the signer certificate will be retrieved. (String, required) **- port** The port of the remote system from where the signer certificate will be retrieved. (Integer, required) **- keyStoreName** The name of the key store where the signer certificate is located. (String, required) **- keyStoreScope** The management scope of the key store. (String, required) **- sslConfigName** The name of the SSL configuration object. (String, optional) **- sslConfigScopeName** The management scope where the SSL configuration object is located. (String, optional) • Returns: The signer certificate is created in the key store file.	**Batch mode example usage:** • Using Jacl: `$AdminTask retrieveSigner FromPort {-host serverHost -port 443 -keyStoreName testKeyStore -certificate Alias serverHostSigner}` • Using Jython string: `AdminTask.retrieveSigner FromPort ('[-host server Host -port 443 -keyStore Name testKeyStore -certi ficateAlias serverHost Signer]')` • Using Jython list: `AdminTask.retrieveSigner FromPort (['-host', 'serverHost', '-port', '443', '-keyStoreName', 'testKeyStore', '-certi ficateAlias', 'serverHost Signer'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask retrieveSigner FromPort {-interactive}` • Using Jython string: `AdminTask.retrieveSigner FromPort ('[-interactive]')` • Using Jython list: `AdminTask.retrieveSigner FromPort (['-interactive'])`

Table 42. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
retrieveSigner InfoFromPort	The **retrieveSigner InfoFromPort** command retrieves signer information from a port on a remote host.	None	• Parameters: **- host** The host name of the system from where the signer certificate will be retrieved. (String, required) **- port** The port of the remote system from where the signer certificate will be retrieved. (Integer, required) **- sslConfigName** The name of the SSL configuration object. (String, optional) **- sslConfigScopeName** The management scope where the SSL configuration object is located. (String, optional) • Returns: Information about the signer certificate from the remote host port.	**Interactive mode example usage:** • Using Jacl: `$AdminTask retrieveSigner InfoFromPort {-interactive}` • Using Jython string: `AdminTask.retrieveSigner InfoFromPort ('[-interactive]')` • Using Jython list: `AdminTask.retrieveSigner InfoFromPort (['-interactive'])`

Commands for the PersonalCertificateCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the personalCertificateCommands group of the AdminTask object:

Table 43.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createSelfSigned Certificate	The **createSelfSigned Certificate** command creates a personal certificate in a key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) **- certificateVersion** The version of the certificate. (String, required) **- certificateSize** The size of the certificate. (Integer, required) **- certificateCommonName** The common name of the certificate. (String, required) **- certificateOrganization** The organization of the certificate. (String, optional) **- certificateOrganizationUnit** The organizational unit of the certificate. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask createSelfSignedCertificate {-keyStoreName testKeyStore -certificateAlias default -certificateCommonName localhost -certificateOrganization ibm}` • Using Jython string: `AdminTask.createSelfSignedCertificate ('[-keyStoreName testKeyStore -certificateAlias default -certificateCommonName localhost -certificateOrganization ibm]')` • Using Jython list: `AdminTask.createSelfSignedCertificate (['-keyStoreName', 'testKeyStore', '-certificateAlias', 'default', '-certificateCommonName', 'localhost', '-certificateOrganization', 'ibm'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createSelfSigned Certificate {-interactive}` • Using Jython string: `AdminTask.createSelfSigned Certificate ('[-interactive]')` • Using Jython list: `AdminTask.createSelfSigned Certificate (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			- certificateLocality The locality of the certificate. (String, optional) **- certificateState** The state of the certificate. (String, optional) **- certificateZip** The zip code of the certificate. (String, optional) **- certificateCountry** The country of the certificate. (String, optional) **- certificateValidDays** The amount of time in days for which the certificate is valid. (Integer, optional) • Returns: None	
deleteCertificate	The **deleteCertificate** command deletes a personal certificate from a key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask deleteCertificate {-interactive}` • Using Jython string: `AdminTask.deleteCertificate ('[-interactive]')` • Using Jython list: `AdminTask.deleteCertificate (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
exportCertificate	The **exportCertificate** command exports a personal certificate from one key store to another.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- keyStorePassword** The password to the key store. (String, required) **- keyFilePath** The full path to a key store file that is located in a file system. The store from where a certificate will be imported or exported. (String, required) **- keyFilePassword** The password to the key store file. (String, required) **- keyFileType** The type of the key file. (String, required) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) **- aliasInKeyStore** (String, optional) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask exportCertificate {-interactive}` • Using Jython string: `AdminTask.exportCertificate ('[-interactive]')` • Using Jython list: `AdminTask.exportCertificate (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
extractCertificate	The **extractCertificate** command extracts the signer part of a personal certificate to a file.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) **- certificateRequestFilePath** The full path of the request file that contains the certificate. (String, required) **- base64Encoded** Set the value of this parameter to `true` if the certificate is ascii base 64 encoded. Set the value of this parameter to `false` if the certificate is binary. (Boolean, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask extractCertificate {-keyStoreName testKeyStore -certificateFilePath c:\temp\ CertFile.arm -certificateAlias testCertificate}` • Using Jython string: `AdminTask.extractCertificate ('[-keyStoreName testKeyStore -certificateFilePath c:\temp\ CertFile.arm -certificateAlias testCertificate]')` • Using Jython list: `AdminTask.extractCertificate (['-keyStoreName', 'testKeyStore', '-certificateFilePath', 'c:\temp\ CertFile.arm', '-certificateAlias', 'testCertificate'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask extractCertificate {-interactive}` • Using Jython string: `AdminTask.extractCertificate ('[-interactive]')` • Using Jython list: `AdminTask.extractCertificate (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getCertificate	The **getCertificate** command obtains information about a particular personal certificate in a key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: Information about the certificate request.	**Interactive mode example usage:** • Using Jacl: `$AdminTask getCertificate {-interactive}` • Using Jython string: `AdminTask.getCertificate ('[-interactive]')` • Using Jython list: `AdminTask.getCertificate (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
importCertificate	The **importCertificate** command imports a personal certificate from a key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- keyFilePath** The full path to a key store file that is located in a file system. The store from where a certificate will be imported or exported. (String, required) **- keyFilePassword** The password to the key store file. (String, required) **- keyFileType** The type of the key file. (String, required) **- certificateAliasFromKeyFile** The certificate alias in the key file from which the certificate is being imported. (String, required) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask importCertificate {-interactive}` • Using Jython string: `AdminTask.importCertificate ('[-interactive]')` • Using Jython list: `AdminTask.importCertificate (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listPersonalCertificates	The **listPersonalCertificates** command lists the personal certificates in a particular key store.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) • Returns: A list of attributes for each personal certificate in a key store.	**Interactive mode example usage:** • Using Jacl: `$AdminTask listPersonalCertificates {-interactive}` • Using Jython string: `AdminTask.listPersonalCertificates ('[-interactive]')` • Using Jython list: `AdminTask.listPersonalCertificates (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
receiveCertificate	The **receiveCertificate** command receives a signer certificate from a file to a personal certificate.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) **- certificateFilePath** The full path of the file that contains the certificate. (String, required) **- base64Encoded** Set the value of this parameter to `true` if the certificate is ascii base 64 encoded. Set the value of this parameter to `false` if the certificate is binary. (Boolean, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask receiveCertificate {-keyStoreName testKeyStore -certificateFilePath c:\temp\ CertFile.arm}` • Using Jython string: `AdminTask.receiveCertificate ('[-keyStoreName testKeyStore -certificateFilePath c:\temp\ CertFile.arm]')` • Using Jython list: `AdminTask.receiveCertificate (['-keyStoreName', 'testKeyStore', '-certificateFilePath', 'c:\ temp\CertFile.arm'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask receiveCertif icate {-interactive}` • Using Jython string: `AdminTask.receiveCerti ficate ('[-interactive]')` • Using Jython list: `AdminTask.receiveCertif icate (['-interactive'])`

Table 43. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
replaceCertificate	The **replaceCertificate** command replaces a personal certificate with a new one. Replaces all signer certificates from the personal certificate.	None	• Parameters: **- keyStoreName** The name that uniquely identifies the key store configuration object. (String, required) **- keyStoreScope** The scope name of the key store. (String, optional) **- certificateAlias** The name that uniquely identifies the certificate request in a key store. (String, required) **- replacementCertificateAlias** The alias of the certificate that is used to replace a different certificate. (String, required) **- deleteOldCert** Set the value of this parameter to `true` if you want to delete the old signer certificates during certificate replacement. Otherwise, set the value of this parameter to `false`. (Boolean, optional) **- deleteOldSigners** Set the value of this parameter to `true` if you want to delete the old certificates during certificate replacement. Otherwise, set the value of this parameter to `false`. (Boolean, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask replaceCertificate {-keyStoreName testKeyStore -certificateAlias default -replacementCertificateAlias replaceCert -deleteOldCert true -deleteOldSigners true}` • Using Jython string: `AdminTask.replaceCertificate ('[-keyStoreName testKeyStore -certificateAlias default -replacementCertificateAlias replaceCert -deleteOldCert true -deleteOldSigners true]')` • Using Jython list: `AdminTask.replaceCertificate (['-keyStoreName', 'testKeyStore', '-certificateAlias', 'default', '-replacementCertificateAlias', 'replaceCert', '-deleteOldCert', 'true', '-deleteOldSigners', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask replaceCertificate {-interactive}` • Using Jython string: `AdminTask.replaceCertificate ('[-interactive]')` • Using Jython list: `AdminTask.replaceCertificate (['-interactive'])`

Commands for the SPNEGO TAI group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SPNEGO TAI group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:
addSpnegoTAIProperties	The **addSpnegoTAIProperties** command adds properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	• Parameters: **- spnId** This is the SPN identifier for the group of custom properties that are to be defined with this command. If you do not specify this parameter, an unused SPN identifier is assigned. (String, optional) **- host** Specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. (String, required) **- filter** Defines the filtering criteria used by the class specified with the above attribute. If no filter is specified, all HTTP requests are subject to SPNEGO authentication. (String, optional) **- filterClass** Specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no filter class is specified, the default filter class, com.ibm.ws.security.spnego.HTTPHeaderFilter, is used. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6` • Using Jython string: `AdminTask.addSpnegoTAIProperties ('[-host myhost.ibm.com -filter user-agent%=IE 6]')` • Using Jython list: `AdminTask.addSpnegoTAIProperties (['-host', 'myhost.ibm.com', '-filter', 'user-agent%=IE', '6'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addSpnegoTAIProperties -interactive` • Using Jython string: `AdminTask.addSpnegoTAIProperties ('[-interactive]')` • Using Jython list: `AdminTask.addSpnegoTAIProperties ['-interactive'])`

			- noSpnegoPage Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. (String, optional). If you do not specify the noSpnegoPage attribute then the default is used: `"<html><head><title> SPNEGO authentication is not supported. </title></head>" + "<body>SPNEGO authe ntication is not supported on this client.</body> </html>";` **- ntlmTokenPage** Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. (String, optional). If you do not specify the ntlmTokenPage attribute then the default is used: `"<html><head><title> An NTLM Token was received.</title> </head>" + "<body>Your bro wser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";`	

			- trimUserName Specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the `@` that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`. (String, optional) • Returns: None	

createKrb ConfigFile	The **createKrb ConfigFile** command creates the Kerberos configuration file for use with the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	• Parameters: **- krbPath** Provides the fully qualified file system location of the Kerberos configuration (krb5.ini or krb5.conf) file. (String, required) **- realm** Provides the Kerberos realm name. The value of this attribute is used by the SPNEGO TAI to form the Kerberos service principal name for each of the hosts specified with the property com.ibm.ws.security.spnego.SPN<id>.hostname (String, required) **- kdcHost** Provides the host name of the Kerberos Key Distribution Center (KDC). (String, required) **- kdcPort** Provides the port number of the KDC. The default value, if not specified, is 88. (String, optional) **- dns** Provides the default domain name service (DNS) that is used to produce a fully qualified host name. (String, required) **- keytabPath** Provides the file system location of the Kerberos keytab file. (String, required) **- encryption** Identifies the list of supported encryption types, separated by a space. The specified value is used for the default_tkt_enctypes and default_tgs_enctypes. The default encryption types, if not specified, are des-cbc-md5 and rc4-hmac. (String, optional) • Returns: None	**Interactive mode example usage:** • Using Jacl: `$AdminTask createKrbCo nfigFile -interactive` • Using Jython string: `AdminTask.createKrbCon figFile ('[-interactive]')` • Using Jython list: `AdminTask.createKrbCon figFile ['-interactive'])`

deleteSpnegoTAIProperties	The **deleteSpnegoTAIProperties** command deletes properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	• Parameters: **- spnId** The SPN identifier for the group of custom properties that are to be deleted with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are deleted. (String, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteSpnegoTAIProperties {-spnId 2}` • Using Jython string: `AdminTask.deleteSpnegoTAIProperties ('[-spnId 2]')` • Using Jython list: `AdminTask.deleteSpnegoTAIProperties (['-spnId', '2'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteSpnegoTAIProperties -interactive` • Using Jython string: `AdminTask.deleteSpnegoTAIProperties ('[-interactive]')` • Using Jython list: `AdminTask.deleteSpnegoTAIProperties ['-interactive'])`

modifySpnegoTAIProperties	The **modifySpnegoTAIProperties** command modifies the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	• Parameters: **- spnId** The SPN identifier for the group of custom properties that are to be defined with this command. (String, required) **- host** Specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. (String, optional) **- filter** Defines the filtering criteria used by the class specified with the above attribute. (String, optional) **- filterClass** Specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no class is specified, all HTTP requests will be subject to SPNEGO authentication. (String, optional) **- noSpnegoPage** Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. (String, optional)	**Batch mode example usage:** • Using Jacl: `$AdminTask modifySpnegoTAI PROPERTIES -spnId 1 -filter host==myhost.company.com` • Using Jython string: `AdminTask.modifySpnegoTAI PROPERTIES ('[-spnId 1 -filter host==myhost.company.com]')` • Using Jython list: `AdminTask.modifySpnegoTAI PROPERTIES (['-spnId', '1', '-filter', 'host==myhost.company.com'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask modifySpnegoTAI Properties -interactive` • Using Jython string: `AdminTask.modifySpnegoTAI Properties ('[-interactive]')` • Using Jython list: `AdminTask.modifySpnegoTAI Properties ['-interactive'])`

			- ntlmTokenPage Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. (String, optional) **- trimUserName** Specifies whether (`true`) or not (`false`) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the ″@″ that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to `false`, the suffix of the principal name is retained. The default value used is `true`. (String, optional) • Returns: None	

showSpnegoTAIProperties	The **showSpnegoTAIProperties** command displays the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	• Parameters: **- spnId** The service principal name (SPN) identifier for the group of custom properties that are to be displayed with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are displayed. (String, optional) • Returns: A list of properties.	**Batch mode example usage:** • Using Jacl: `$AdminTask showSpnegoTAI Properties -spnId 1` • Using Jython string: `AdminTask.showSpnegoTAI Properties ('[-spnId 1]')` • Using Jython list: `AdminTask.showSpnegoTAI Properties (['-spnId', '1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask showSpnegoTAI Properties -interactive` • Using Jython string: `AdminTask.showSpnegoTAI Properties ('[-interact ive]')` • Using Jython list: `AdminTask.showSpnegoTAI Properties ['-interact ive'])`

Commands for the AuthorizationGroupCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the AuthorizationGroupCommands group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:

addResourceToAuthorizationGroup	The **addResourceToAuthorizationGroup** command adds a resource instance to an existing authorization group. A resource instance cannot belong to more than one authorization group.	None	• Parameters: **- authorizationGroupName** The name of the authorization group. (String, required) **- resourceName** The name of the resource instance that you want to add to an authorization group. (String, required) The resourceName parameter should be in the following format: `ResourceType= ResourceName` where `ResourceType` is one of the following values: Application, Server, ServerCluster, Node, NodeGroup `ResourceName` is the name of the resource instance, for example, `server1`. The following are example uses of the resourceName parameter: – `Node=node1:Se rver=server1` This example uniquely identifies `server1`. `node1` is required if another `server1` exists on a different node. – `Application=app1` • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask addResourceToAuthorizationGroup {-authorizationGroupName groupName -resourceName Application=app1}` • Using Jython string: `AdminTask.addResourceToAuthorizationGroup('[-authorizationGroupName groupName -resourceName Application=app1]')` • Using Jython list: `AdminTask.addResourceToAuthorizationGroup(['-authorizationGroupName', 'groupName', '-resourceName', 'Application=app1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask addResourceToAuthorizationGroup {-interactive}` • Using Jython string: `AdminTask.addResourceToAuthorizationGroup ('[-interactive]')` • Using Jython list: `AdminTask.addResourceToAuthorizationGroup (['-interactive'])`

createAuthorizationGroup	The **createAuthorizationGroup** command creates a new authoirzation group. When you create a new authorization group, no members are associated with it. Also, no user to administrative role mapping for the authorization table is associated with the authorization group.	None	• Parameters: **- authorizationGroupName** The name of the authorization group that you want to create. (String, required) • Returns: The configuration ID of the authorization group that you created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createAuthorizationGroup {-authorizationGroupName groupName}` • Using Jython string: `AdminTask.createAuthorizationGroup('[-authorizationGroupName groupName]')` • Using Jython list: `AdminTask.createAuthorizationGroup(['-authorizationGroupName', 'groupName'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask createAuthorizationGroup -interactive` • Using Jython string: `AdminTask.createAuthorizationGroup ('[-interactive]')` • Using Jython list: `AdminTask.createAuthorizationGroup (['-interactive'])`

deleteAuthorizationGroup	The deletes an existing authorization group. When you delete an authorization group, the authorization table that corresponds is also deleted.	None	• Parameters: **- authorizationGroup Name** The name of the authorization group that you want to delete. (String, required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteAuthorizationGroup {-authorizationGroupName groupName}` • Using Jython string: `AdminTask.deleteAuthorizationGroup('[-authorizationGroupName groupName]')` • Using Jython list: `AdminTask.deleteAuthorizationGroup(['-authorizationGroupName', 'groupName'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteAuthorizationGroup {-interactive}` • Using Jython string: `AdminTask.deleteAuthorizationGroup ('[-interactive]')` • Using Jython list: `AdminTask.deleteAuthorizationGroup (['-interactive'])`
	The command lists the existing authorization groups.	None	• Parameters: None • Returns: A list of short names of all existing authorization groups. (String [])	**Batch mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroups` • Using Jython: `AdminTask.listAuthorizationGroups()` **Interactive mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroups {-interactive}` • Using Jython string: `AdminTask.listAuthorizationGroups ('[-interactive]')` • Using Jython list: `AdminTask.listAuthorizationGroups (['-interactive'])`

listAuthorizationGroupsForGroupID	The **listAuthorizationGroupsForGroupID** command lists all of the authorization groups to which a given user group has access. This command lists the authorization groups and the granted roles for each authorization group. The group ID can be a short name or a fully qualified domain name if the LDAP user registry is being used. This command will list `cell` as a group if the user has cell level access.	None	• Parameters: **- groupid** The ID of the user group. (String, required) • Returns: The map of the authorization group and granted roles. (Map[String=String[]])	**Batch mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroupsForGroupID {-groupid userGroupName}` • Using Jython string: `AdminTask.listAuthorizationGroupsForGroupID('[-groupid userGroupName]')` • Using Jython list: `AdminTask.listAuthorizationGroupsForGroupID(['-groupid', 'userGroupName'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroupsForGroupID {-interactive}` • Using Jython string: `AdminTask.listAuthorizationGroupsForGroupID ('[-interactive]')` • Using Jython list: `AdminTask.listAuthorizationGroupsForGroupID (['-interactive'])`

listAuthorizationGroupsForUser ID	The **listAuthorizationGroupsForUser ID** command lists all of the authorization groups to which a given user has access. This command lists the authorization groups and the granted roles for each authorization group. The user ID and the group ID can be a short name or a fully qualified domain name if the LDAP user registry is being used. This command will list `cell` as a group if the user has cell level access.	None	• Parameters: **- userid** The ID of the user. (String, required) • Returns: The map of the authorization group and granted roles. (Map[String=String[]])	**Batch mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroupsForUserID{-userid userName}` • Using Jython string: `AdminTask.listAuthorizationGroupsForUserID('[-userid userName]')` • Using Jython list: `AdminTask.listAuthorizationGroupsForUserID(['-userid', 'userName'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroupsForUserID {-interactive}` • Using Jython string: `AdminTask.listAuthorizationGroupsForUserID ('[-interactive]')` • Using Jython list: `AdminTask.listAuthorizationGroupsForUserID (['-interactive'])`

listAuthorizationGroupsOfResource	The **listAuthorizationGroupsOfResource** command lists authorization groups for a given resource. If the value of the traverseContainedObjects parameter is false, only the authorization group of the resource is returned. If the value of the traverseContainedObjects parameter is true, it returns the authorization group of the resource and the authorization groups of all the parent resources in the containment tree.	None	• Parameters: **- resourceName** The name of the resource. (String, required) The resourceName parameter must be in the following format: `ResourceType=ResourceName` where `ResourceType` can be any one of the following values: Application, Server, ServerCluster, Node, or NodeGroup. `ResourceName` is the name of the resource instance, for example, server1. The following are examples of the resourceName parameter: `Node=node1: Server=server` This example uniquely identifies *server1*. The name of the node is required if a server on a different node uses the same server name. `Application=app1`	**Batch mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroupsOfResource {-resourceName Application=app1}` • Using Jython string: `AdminTask.listAuthorizationGroupsOfResource('[-resourceName Application=app1]')` • Using Jython list: `AdminTask.listAuthorizationGroupsOfResource(['-resourceName', 'Application=app1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listAuthorizationGroupsOfResource {-interactive}` • Using Jython string: `AdminTask.listAuthorizationGroupsOfResource ('[-interactive]')` • Using Jython list: `AdminTask.listAuthorizationGroupsOfResource (['-interactive'])`

			- traverseContained Resources Finds the authorization groups of all the parent resources by traversing the resource containment tree upwards. The default value is false. (Boolean, optional) • Returns: The short names of all of the authorization groups for which the resource belongs. If the you do not specify the traverseContainedResources parameter, the result of this command will only contain one value because a resource instance can only belong to one authorization group. (String[])	
listResourcesOfAuthorizationGroup	The **listResourcesOfAuthorizationGroup** command lists all of the resources within the given authorization group.	None	• Parameters: **-authorizationGroupName** The name of the authorization group. (String, required) • Returns: The configuration IDs of all of the resource instances within the authorization group. (String[])	**Batch mode example usage:** • Using Jacl: `$AdminTask listResourcesOfAuthorizationGroup {-authorizationGroupName groupName}` • Using Jython string: `AdminTask.listResourcesOfAuthorizationGroup('[-authorizationGroupName groupName]')` • Using Jython list: `AdminTask.listResourcesOfAuthorizationGroup(['-authorizationGroupName', 'groupName'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listResourcesOfAuthorizationGroup {-interactive}` • Using Jython string: `AdminTask.listResourcesOfAuthorizationGroup ('[-interactive]')` • Using Jython list: `AdminTask.listResourcesOfAuthorizationGroup (['-interactive'])`

listResources ForGroupID	The **listResources ForGroupID** command lists all the objects that a given group has access to. This command lists the resources and the granted roles for each resource. The resources that this command returns include the resources from the authorization groups to which the user group is granted roles and the resources that are descendants of the resources with in authorization groups to which the user group is granted access to any role. The group ID can be a short name or fully qualified domain name if a LDAP user registry is used.	None	• Parameters: **- groupid** The ID of the user group. (String, required) • Returns: The map of the granted role and resources. (Map[String=String[]])	**Batch mode example usage:** • Using Jacl: `$AdminTask listResourcesFor GroupID {-groupid userGr oupName}` • Using Jython string: `AdminTask.listResourcesFor GroupID('[-groupid userGro upName]')` • Using Jython list: `AdminTask.listResourcesFor GroupID(['-groupid', 'user GroupName'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listResourcesFor GroupID {-interactive}` • Using Jython string: `AdminTask.listResourcesFor GroupID ('[-interactive]')` • Using Jython list: `AdminTask.listResourcesFor GroupID (['-interactive'])`

listResources ForUserID	The **listResources ForUserID** command lists all the objects that a given user has access to. This command lists the resources and the granted roles for each resource. The resources that this command returns include the resources from the authorization groups to which the user is granted roles and the resources that are descendants of the resources with in authorization groups to which the user is granted access to any role. The user ID can be a short name or fully qualified domain name if a LDAP user registry is used.	None	• Parameters: **- userid** The ID of the user. (String, required). • Returns: The map of granted role and resources. (Map[String=String[]])	**Batch mode example usage:** • Using Jacl: `$AdminTask listResourcesFor UserID {-userid userName }` • Using Jython string: `AdminTask.listResourcesFor UserID('[-userid userName]')` • Using Jython list: `AdminTask.listResourcesFor UserID(['-userid', 'userName'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listResources ForUserID {-interactive}` • Using Jython string: `AdminTask.listResources ForUserID ('[-interactive]')` • Using Jython list: `AdminTask.listResources ForUserID (['-interactive'])` **Example output:** `{deployer=[], operator=[], administrator=[cells/IBM-LP1 6L31HVE8Cell07/clusters/C1|cluster.xml, cells/IBM-LP16L 31HVE8Cell07/nodes/IBM-LP16L 31HVE8Node05/servers/cm1|server.xml], monitor=[], configurator=[]}`

mapGroupsToAdminRole	The **mapGroupsToAdminRole** command maps group IDs to one or more administrative roles in an authorization group. The name of the authorization group that you provide determines which authorization table will be used. If you do not specify an authorization group name, the mapping is done to the cell level authorization table. The group ID can be a short name or a fully qualified domain name if the LDAP user registry is used.	None	• Parameters: **- authorizationGroupName** The name of the authorization group. If you do not specify this parameters, the cell level authorization group is assumed. (String, optional) **- roleName** The name of the administrative role. (String, required) **- groupids** The list of group IDs that will mapped to the administrative role. (String[], required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask mapGroupsToAdminRole {-authorizationGroupName group Name - roleName administrator -groupids group1}` • Using Jython string: `AdminTask.mapGroupsToAdminRole ('[-authorizationGroupName group Name -roleName administrator -groupids group1]')` • Using Jython list: `AdminTask.mapGroupsToAdminRole (['-authorizationGroupName', 'groupName', '-roleName', 'administrator', '-groupids', 'group1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask mapGroupsToAdm inRole {-interactive}` • Using Jython string: `AdminTask.mapGroupsToAdmin Role ('[-interactive]')` • Using Jython list: `AdminTask.mapGroupsToAdmin Role (['-interactive'])`

mapUsersTo AdminRole	The **mapUsersTo AdminRole** command maps user IDs to one or more administrative roles in the authorization group. The name of the authorization group that you provide determines the authorization table. If you do not specify the name of the authorization group, the mapping is done to the cell level authorization table. The user ID can be a short name or fully qualified domain name in case LDAP user registry is used.	None	• Parameters: **- authorizationGroup Name** The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional) **- roleName** The name of the administrative role. (String, required) **- userids** The list of user IDs that will be mapped to the administrative role (String[], required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask mapUsersToAdminRole {-authorizationGroupName group Name - roleName administrator -userids user1}` • Using Jython string: `AdminTask.mapUsersToAdminRole ('[-authorizationGroupName groupName -roleName administ rator -userids user1]')` • Using Jython list: `AdminTask.mapUsersToAdminRole (['-authorizationGroupName', 'groupName', '-roleName', 'administrator', '-userids', 'user1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask mapUsersToAdmin Role {-interactive}` • Using Jython string: `AdminTask.mapUsersToAdmin Role ('[-interactive]')` • Using Jython list: `AdminTask.mapUsersToAdmin Role (['-interactive'])`

removeGroupsFromAdminRole	The **removeGroupsFromAdminRole** command removes previously mapped group IDs from administrative roles in the authorization group. The name of the authorization group that you provide determines which authorization table is involved. If you do not specify an authorization group name, the group IDs are removed from the cell level authorization table. The group ID can be a short name or fully qualified domain name if a LDAP user registry is used.	None	• Parameters: **- authorizationGroupName** The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional) **- roleName** The name of the administrative role. (String, required) **- userids** A list of group IDs that you want to remove from the administrative role. (String[], required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask removeGroupsFromAdminRole {-authorizationGroupName groupName - roleName administrator -groupids group1}` • Using Jython string: `AdminTask.removeGroupsFromAdminRole('[-authorizationGroupName groupName -roleName administrator -groupids group1]')` • Using Jython list: `AdminTask.removeGroupsFromAdminRole(['-authorizationGroupName', 'groupName', '-roleName', 'administrator', '-groupids', 'group1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeGroupsFromAdminRole {-interactive}` • Using Jython string: `AdminTask.removeGroupsFromAdminRole ('[-interactive]')` • Using Jython list: `AdminTask.removeGroupsFromAdminRole (['-interactive'])`

removeResourceFromAuthorizationGroup	The **removeResourceFromAuthorizationGroup** command removes resources from an existing authorization group. If you do not specify the authorization group, it will be determined and the resource will be removed from that authorization group.	None	• Parameters: **- authorizationGroupName** The name of the authorization group. (String, optional) **- resourceName** The name of the resource instance that you want to remove from the authorization group. (String, required) The resourceName parameter must be in the following format: `ResourceType=ResourceName` where the `ResourceType` can be any of the following: Application, Server, ServerCluster, Node, or NodeGroup. The `ResourceName` is the name of the resource instance, for example, server1. The following are examples of the resourceName parameter: `Node=node1:Server=server1` This example uniquely identifies server1. node1 is required if the name of the server exists on multiple nodes. `Application=app1` • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask removeResourceFromAuthorizationGroup {-authorizationGroupName groupName -resourceName Application=app1}` • Using Jython string: `AdminTask.removeResourceFromAuthorizationGroup('[-authorizationGroupName groupName -resourceName Application=app1]')` • Using Jython list: `AdminTask.removeResourceFromAuthorizationGroup(['-authorizationGroupName', 'groupName', '-resourceName', 'Application=app1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeResourceFromAuthorizationGroup {-interactive}` • Using Jython string: `AdminTask.removeResourceFromAuthorizationGroup ('[-interactive]')` • Using Jython list: `AdminTask.removeResourceFromAuthorizationGroup (['-interactive'])`

removeUsers FromAdmin Role	The **removeUsers FromAdmin Role** command removes previously mapped user IDs from administrative roles in the authorization group. The name of the authorization group that you provide determines which authorization table is involved. If you do not specify an authorization group name, the user ID from the cell level authorization table will be used. The user ID can be a short name or a fully qualified domain name if a LDAP user registry is used.	None	• Parameters: **- authorizationGroup Name** The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional) **- roleName** The name of the administrative role. (String, required) **- userids** A list of user IDs that you want to remove from the administrative role. (String[], required) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask removeUsersFromAdmin Role {-authorizationGroupName groupName - roleName administra tor -userids user1}` • Using Jython string: `AdminTask.removeUsersFromAdmin Role('[-authorizationGroupName groupName -roleName administra tor -userids user1]')` • Using Jython list: `AdminTask.removeUsersFromAdmin Role(['-authorizationGroupName', 'groupName', '-roleName', 'adm inistrator', '-userids', 'user1'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask removeUsersFrom AdminRole {-interactive}` • Using Jython string: `AdminTask.removeUsersFrom AdminRole ('[-interactive]')` • Using Jython list: `AdminTask.removeUsersFrom AdminRole (['-interactive'])`

Commands for the ChannelFrameworkManagement group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the ChannelFrameworkManagement group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:

createChain	The **createChain** command creates a new chain of transport channels that are based on a chain template.	The instance of the transport channel service under which the new chain is created. (ObjectName, required)	• Parameters: **- template** The chain template on which to base the new chain. (ObjectName, required) **- name** The name of the new chain. (String, required) **- endPoint** The name of the end point to be used by the instance of the TCP inbound channel in the new chain if the chain is an inbound chain. (ObjectName, optional) • Returns: The object name of the channel chain that was created.	**Batch mode example usage:** • Using Jacl: `$AdminTask createChain (cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr\|server.xml#TransportChannelService_1) {-template WebContainer(templates/chains\|webcontainer-chains.xml#Chain_1) -name trialChain1}` `$AdminTask createChain (cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr|server.xml#TransportChannelService_1) {-template WebContainer(templates/chains|webcontainer-chains.xml#Chain_1) -name trialChain1 -endPoint (cells/rohitbuildCell01/nodes/rohitbuildCellManager01|serverindex.xml#EndPoint_3) }` • Using Jython string: `AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr|server.xml#TransportChannelService_1', '[-template "WebContainer(templates/chains|webcontainer-chains.xml#Chain_1)" -name trialChain]')` `AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr|server.xml#TransportChannelService_1', '[-template "WebContainer(templates/chains|webcontainer-chains.xml#Chain_1)" -name trialChain -endPoint "(cells/rohitbuildCell01/nodes/rohitbuildCellManager01|serverindex.xml#EndPoint_3)"]')` • Using Jython list: `AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr|server.xml#TransportChannelService_1', ['-template', "WebContainer(templates/chains|webcontainer-chains.xml#Chain_1)", '-name', 'trialChain'])` `AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr|server.xml#TransportChannelService_1', ['-template', "WebContainer(templates/chains|webcontainer-chains.xml#Chain_1)", '-name', 'trialChain', '-endPoint', "(cells/rohitbuildCell01/nodes/rohitbuildCellManager01|serverindex.xml#EndPoint_3)"])`

				Interactive mode example usage: • Using Jacl: `$AdminTask createChain {-interactive}` • Using Jython string: `AdminTask.createChain ('[-interactive]')` • Using Jython list: `AdminTask.createChain (['-interactive'])`
deleteChain	The **deleteChain** command deletes an existing chain and, optionally, the transport channels in the chain.	The chain to be deleted. (Object name, required)	• Parameters: **- deleteChannels** If the value of this attribute is true, non-shared transport channels used by the specified chain will be deleted. (Boolean, optional) • Returns: None	**Batch mode example usage:** • Using Jacl: `$AdminTask deleteChain trialChain1 (cells/rohitbuildCell01/nodes/roh itbuildCellManager01/servers/dmgr\|server.xml#Chain_1093554462922)` `$AdminTask deleteChain trialChain (cells/rohitbuildCell01/nodes/roh itbuildCellManager01/servers/dmgr |server.xml#Chain_1093554378078) {-deleteChannels true}` • Using Jython string: `AdminTask.deleteChain('trialChain 1(cells/rohitbuildCell01/nodes/roh itbuildCellManager01/servers/dmgr| server.xml#TransportChannelSer vice_1)')` `AdminTask.deleteChain('trialChain1 (cells/rohitbuildCell01/nodes/rohi tbuildCellManager01/servers/dmgr| server.xml#TransportChannelService _1)', '[-deleteChannels true]')` • Using Jython list: `AdminTask.deleteChain('trialChain1 (cells/rohitbuildCell01/nodes/roh itbuildCellManager01/servers/dmgr| server.xml#TransportChannelService _1)')` `AdminTask.deleteChain('trialChain1 (cells/rohitbuildCell01/nodes/rohi tbuildCellManager01/servers/dmgr| server.xml#TransportChannelServic e_1)', ['-deleteChannels', 'true'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask deleteChain {-interactive}` • Using Jython string: `AdminTask.deleteChain ('[-interactive]')` • Using Jython list: `AdminTask.deleteChain (['-interactive'])`

listChain Templates	The **listChain Templates** command displays a list of templates that you can use to create chains in this configuration. All templates have a certain type of transport channel as the last transport channel in the chain.	None	• Parameters: **- acceptorFilter** The templates returned by this method all have a transport channel instance of the specified type as the last transport channel in the chain. (String, optional) • Returns: A list of all the chain template object names. If you specify the acceptorFilter parameter, the list that returns is filtered to match the filter that you specified.	**Batch mode example usage:** • Using Jacl: `$AdminTask listChainTemplates {}` `$AdminTask listChainTemplates "-acceptorFilter WebContainer InboundChannel"` • Using Jython string: `AdminTask.listChainTemplates()` `AdminTask.listChainTemplates ('[-acceptorFilter WebCont ainerInboundChannel]')` • Using Jython list: `AdminTask.listChainTemplates()` `AdminTask.listChainTemplates (['-acceptorFilter', 'WebC ontainerInboundChannel'])` **Interactive mode example usage:** • Using Jacl: `$AdminTask listChainTemplates {-interactive}` • Using Jython string: `AdminTask.listChainTemplates ('[-interactive]')` • Using Jython list: `AdminTask.listChainTemplates (['-interactive'])`

listChains	The **listChains** command lists all the chains that are configured under a particular instance of the transport channel service.	The instance of the transport channel service under which the chains are configured. (ObjectName, required)	• Parameters: **- acceptorFilter** The chains that are returned by this parameter will have a transport channel instance of the type that you specify as the last transport channel in the chain. (String, optional) **- endPointFilter:** The chains returned by this parameter will have a TCP inbound channel using an end point with the name that you specify.(String, optional) • Returns: A list of all the channel chain object names that match the specified filters. If no you do not specify any parameters, all of the channel chains that are configured under the particular instance of transport channel service are returned.	**Batch mode example usage:** • Using Jacl: `$AdminTask listChains (cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2\|server.xml#TransportChannelService_1093445762328)` `$AdminTask listChains (cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328) {-acceptorFilter *WebContainerInboundChannel*}` `$AdminTask listChains (cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328) {-endPointFilter *WC_adminhost*}` • Using Jython string: `AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)')` `AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)', '[-acceptorFilter WebContainerInboundChannel]')` `AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)', '[-endPointFilter WC_adminhost]')` • Using Jython list: `AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)')` `AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)', ['-acceptorFilter', 'WebContainerInboundChannel'])` `AdminTask.listChains('(cells/rohitbuildCell01/nodes/rohitbuildNode01/servers/server2|server.xml#TransportChannelService_1093445762328)', ['-endPointFilter', 'WC_adminhost'])`

				Interactive mode example usage: • Using Jacl: `$AdminTask listChains {-interactive}` • Using Jython string: `AdminTask.listChains ('[-interactive]')` • Using Jython list: `AdminTask.listChains (['-interactive'])`

Chapter 17. Naming and directory

Using naming

Naming is used by clients of WebSphere Application Server applications most commonly to obtain references to objects related to those applications, such as Enterprise JavaBeans (EJB) homes.

The Naming service is based on the Java Naming and Directory Interface (JNDI) 1.2.1 Specification and the Object Management Group (OMG) Interoperable Naming (CosNaming) specifications Naming Service Specification, Interoperable Naming Service revised chapters and Common Object Request Broker: Architecture and Specification (CORBA).

1. Develop your application using either JNDI or CORBA CosNaming interfaces. Use these interfaces to look up server application objects that are bound into the name space and obtain references to them. Most Java developers use the JNDI interface. However, the CORBA CosNaming interface is also available for performing Naming operations on WebSphere Application Server name servers or other CosNaming name servers.
2. Assemble your application using an assembly tool. Application assembly is a packaging and configuration step that is a prerequisite to application deployment. If the application you are assembling is a client to an application running in another process, you should qualify the jndiName values in the deployment descriptors for the objects related to the other application. Otherwise, you may need to override the names with qualified names during application deployment. If the objects have fixed qualified names configured for them, you should use them so that the jndiName values do not depend on the other application's location within the topology of the cell.
3. **Optional:** Verify that your application is assigned the appropriate security role if administrative security is enabled. For more information on the security roles, see "Naming roles" on page 1780.
4. Deploy your application. Put your assembled application onto the application server. If the application you are assembling is a client to an application running in another server process, be sure to qualify the jndiName values for the other application's server objects if they are not already qualified. For more information on qualified names, refer to "Lookup names support in deployment descriptors and thin clients" on page 1775.
5. Configure name space bindings. This step is necessary in these cases:
   - Your deployed application is to be accessed by legacy client applications running on previous versions of WebSphere Application Server. In this case, you must configure additional name bindings for application objects relative to the default initial context for legacy clients. (Version 5 clients have a different initial context from legacy clients.)
   - The application requires qualified name bindings for such reasons as:
     – It will be accessed by J2EE client applications or server applications running in another server process.
     – It will be accessed by thin client applications.

     In this case, you can configure name bindings as additional bindings for application objects. The qualified names for the configured bindings are *fixed*, meaning they do not contain elements of the cell topology that can change if the application is moved to another server. Objects as bound into the name space by the system can always be qualified with a topology-based name. You must explicitly configure a name binding to use as a fixed qualified name.

   For more information on qualified names, refer to "Lookup names support in deployment descriptors and thin clients" on page 1775. For more information on configured name bindings, refer to "Configured name bindings" on page 1777.
6. Troubleshoot any problems that develop. If a Naming operation is failing and you need to verify whether certain name bindings exist, use the dumpNameSpace tool to generate a dump of the name space.

Naming

Naming is used by clients of WebSphere Application Server applications to obtain references to objects related to those applications, such as enterprise bean (EJB) homes.

These objects are bound into a mostly hierarchical structure, referred to as a *name space*. In this structure, all non-leaf objects are called *contexts*. Leaf objects can be contexts and other types of objects. Naming operations, such as lookups and binds, are performed on contexts. All naming operations begin with obtaining an *initial context*. You can view the initial context as a starting point in the name space.

The name space structure consists of a set of *name bindings*, each consisting of a name relative to a specific context and the object bound with that name. For example, the name `myApp/myEJB` consists of one non-leaf binding with the name `myApp`, which is a context. The name also includes one leaf binding with the name `myEJB`, relative to `myApp`. The object bound with the name `myEJB` in this example happens to be an EJB home reference. The whole name `myApp/myEJB` is relative to the initial context, which you can view as a starting place when performing naming operations.

You can access and manipulate the name space through a *name server*. Users of a name server are referred to as *naming clients*. Naming clients typically use the Java Naming and Directory Interface (JNDI) to perform naming operations. Naming clients can also use the Common Object Request Broker Architecture (CORBA) CosNaming interface.

You can use security to control access to the name space. For more information, see Naming roles.

Typically, objects bound to the name space are resources and objects associated with installed applications. These objects are bound by the system, and client applications perform lookup operations to obtain references to them. Occasionally, server and client applications bind objects to the name space. An application can bind objects to transient or persistent partitions, depending on requirements.

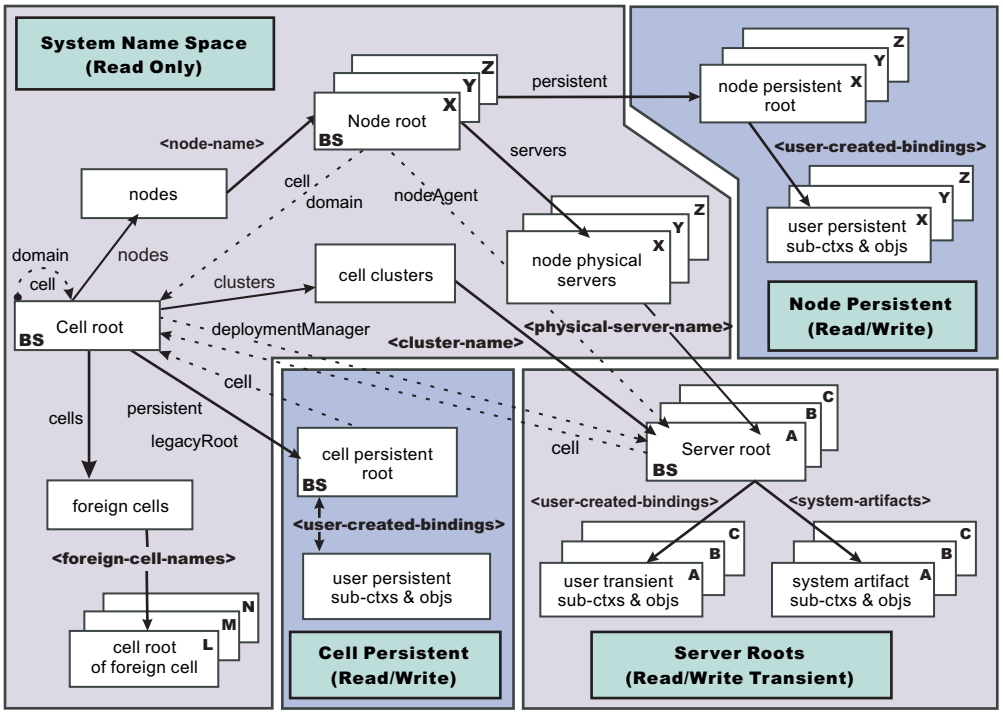
In J2EE environments, some JNDI operations are performed with `java:` URL names. Names bound under these names are bound to a completely different name space which is local to the calling process. However, some lookups on the `java:` name space may trigger indirect lookups to the name server.

Name space logical view

The name space for the entire cell is federated among all servers in the cell. Every server process contains a name server. All name servers provide the same logical view of the cell name space.

The various server roots and persistent partitions of the name space are interconnected by a system name space. You can use the system name space structure to traverse to any context in a the cell's name space. A logical view of the name space is shown in the following diagram.

Logical View of a Cell's Name Space



The bindings in the preceding diagram appear with solid arrows, labeled in bold, and dashed arrows, labeled in gray. Solid arrows represent *primary bindings*. A primary binding is formed when the associated subcontext is created. Dashed arrows show *linked bindings*. A linked binding is formed when an existing context is bound under an additional name. Linked bindings are added for convenience or interoperability with previous WebSphere Application Server versions.

A cell name space is composed of contexts which reside in servers throughout the cell. All name servers in the cell provide the same logical view of the cell name space. A name server constructs this view at startup by reading configuration information. Each name server has its own local in-memory copy of the name space and does not require another running server to function. There are, however, a few exceptions. Server roots for other servers are not replicated among all the servers. The respective server for a server root must be running to access that server root context.

Name space partitions

There are four major partitions in a cell name space:
- System name space partition
- Server roots partition
- Cell persistent partition
- Node persistent partition

System name space partition

The system name space contains a structure of contexts based on the cell topology. The system structure supports traversal to all parts of a cell name space and to the cell root of other cells, which are configured as foreign cells. The root of this structure is the cell root. In addition to the cell root, the system structure contains a node root for each node in the cell. You can access other contexts of interest specific to a node from the node root, such as the node persistent root and server roots for servers configured in that node.

All contexts in the system name space are read-only. You cannot add, update, or remove any bindings.

Server roots partition

Each server in a cell has a server root context. A server root is specific to a particular server. You can view the server roots for all servers in a cell as being in a transient read/write partition of the cell name space. System artifacts, such as EJB homes for server applications and resources, are bound under the server root context of the associated server. A server application can also add bindings under its server root. These bindings are transient. Therefore, the server application creates all required bindings at application startup, so they exist anytime the application is running.

A server cluster is composed of many servers that are logically equivalent. Each member of the cluster has its own server root. These server roots are not replicated across the cluster. In other words, adding a binding to the server root of one member does not propagate it to the server roots of the other cluster members. To maintain the same view across the cluster, you should create all user bindings under the server root by the server application at application startup so that the bindings are present under the server root of each cluster member. Because of Workload Management (WLM) behavior, a JNDI client outside a cluster has no control over which cluster member's server root context becomes the target of the JNDI operation. Therefore, you should execute bind operations to the server root of a cluster member from within that cluster member process only.

Server-scoped configured name bindings are relative to a server's server root.

Cell persistent partition

The root context of the cell persistent partition is the cell persistent root. A binding created under the cell persistent root is saved as part of the cell configuration and continues to exist until it is explicitly removed. Applications that need to create additional persistent bindings of objects generally associated with the cell can bind these objects under the cell persistent root.

It is important to note that the cell persistent area is not designed for transient, rapidly changing bindings. The bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Cell-scoped configured name bindings are relative to a cell's cell persistent root.

Node persistent partition

The node persistent partition is similar to the cell partition except that each node has its own node persistent root. A binding created under a node persistent root is saved as part of that node configuration and continues to exist until it is explicitly removed.

Applications that need to create additional persistent bindings of objects associated with a specific node can bind those objects under that particular node's node persistent root. As with the cell persistent area, it is important to note that the node persistent area is not designed for transient, rapidly changing bindings. These bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Node-scoped configured name bindings are relative to a node's node persistent root.

Initial context support

All naming operations begin with obtaining an initial context. You can view the initial context as a starting point in the name space. Use the initial context to perform naming operations, such as looking up and binding objects in the name space.

Initial contexts registered with the ORB as initial references

The server root, cell persistent root, cell root, and node root are registered with the name server's ORB and can be used as an initial context. An initial context is used by CORBA and enterprise bean applications as a starting point for name space lookups. The keys for these roots as recognized by the ORB are shown in the following table:

Root Context	Initial Reference Key

Server Root	NameServiceServerRoot
Cell Persistent Root	NameServiceCellPersistentRoot
Cell Root	NameServiceCellRoot, NameService
Node Root	NameServiceNodeRoot

A server root initial context is the server root context for the specific server you are accessing. Similarly, a node root initial context is the node root for the server being accessed.

You can use the previously mentioned keys in CORBA INS object URLs (corbaloc and corbaname) and as an argument to an ORB `resolve_initial_references` call. For examples, see CORBA and JNDI programming examples, which show how to get an initial context.

Default initial contexts

The default initial context depends on the type of client. Different categories of clients and the corresponding default initial context follow.

- **WebSphere Application Server V5 and later JNDI interface implementation**

  The JNDI interface is used by EJB applications to perform name space lookups. WebSphere Application Server clients by default use the WebSphere Application Server CosNaming JNDI plug-in implementation. The default initial context for clients of this type is the server root of the server specified by the provider URL. For more details, refer to the JNDI programming examples on getting initial contexts.
- **Other JNDI implementation**

  Some applications can perform name space lookups with a non-WebSphere Application Server CosNaming JNDI plug-in implementation. Assuming the key **`NameService`** is used to obtain the initial context, the default initial context for clients of this type is the cell root.
- **CORBA**

  The standard CORBA client obtains an initial org.omg.CosNaming.NamingContext reference with the key **`NameService`**. The initial context in this case is the cell root.

Lookup names support in deployment descriptors and thin clients

Server application objects, such as EJB homes, are bound relative to the server root context for the server in which the application is installed. Other objects, such as resources, can also be bound to a specific server root. The names used to look up these objects must be qualified so as to select the correct server root. This topic discusses what relative and qualified names are, when they can be used, and how you can construct them.

Relative names

All names are relative to a context. Therefore, a name that can be resolved from one context in the name space cannot necessarily be resolved from another context in the name space. This point is significant because the system binds objects with names relative to the server root context of the server in which the application is installed. Each server has its own server root context. The initial JNDI context is by default the server root context for the server identified by the provider URL used to obtain the initial context. (Typically, the URL consists of a host and port.) For applications running in a server process, the default initial JNDI context is the server root for that server. A relative name will resolve successfully when the initial context is obtained from the server which contains the target object, but it will not resolve successfully from an initial context obtained from another server.

If all clients of a server application run in the same server process as the application, all objects associated with that application are bound to the same initial context as the clients' initial context. In this case, only names relative to the server's server root context are required to access these server objects.

Frequently, however, a server application has clients that run outside the application's server process. The initial context for these clients can be different from the server application's initial context, and lookups on the relative names for server objects may fail. These clients need to use the qualified name for the server objects. This point must be considered when setting up the jndiName values in a J2EE client application deployment descriptors and when constructing lookup names in thin clients. Qualified names resolve successfully from any initial context in the cell.

Qualified names

All names are relative to a context. Here, the term *qualified name* refers to names that can be resolved from any initial context in a cell. This action is accomplished by using names that navigate to the same context, the cell root. The rest of the qualified name is then relative to the cell root and uniquely identifies an object throughout the cell. All initial contexts in a server (that is, all naming contexts in a server registered with the ORB as an initial reference) contain a binding with the name **cell**, which links back to the cell root context. All qualified names begin with the string **cell/** to navigate from the current initial context back to the cell root context.

A qualified name for an object is the same throughout the cell. The name can be topology-based, or some fixed name bound under the cell persistent root. Topology-based names, described in more detail below, navigate through the system name space to reach the target object. A fixed name bound under the cell persistent root has the same qualified name throughout the cell and is independent of the topology. Creating a fixed name under the cell persistent root for a server application object requires an extra step when the server application is installed, but this step eliminates impacts to clients when the application is moved to a different location in the cell topology. The process for creating a fixed name is described later in this section.

Generally, you **must** use qualified names for EJB jndiName values in a J2EE client application deployment descriptors and for EJB lookup names in thin clients. The only exception is when the initial context is obtained from the server in which the target object resides. For example, a session bean which is a client to an entity bean can use a relative name if the two beans run in the same server. If the session bean and entity beans run in different servers, the jndiName for the entity bean must be qualified in the session bean's deployment descriptors. The same requirement may be true for resources as well, depending on the scope of the resource.

- **Topology-based names**

  The system name space partition in a cell's name space reflects the cell's topology. This structure can be navigated to reach any object bound into the cell's name space. Topology-based qualified names include elements from the topology which reflect the object's location within the cell. For a system-bound object, such as an EJB home, the form for a topology-based qualified name depends on whether the object is bound to a single server or cluster. Both forms are described below.

  Single server

  An object bound in a single server has a topology-based qualified name of the following form:

  `cell/nodes/`*`nodeName`*`/servers/`*`serverName`*`/`*`relativeJndiName`*

  where *nodeName* and *serverName* are the node name and server name for the server where the object is bound, and *relativeJndiName* is the unqualified name of the object; that is, the object's name relative to its server's server root context.

  Server cluster

  An object bound in a server cluster has a topology-based qualified name of the following form:

  `cell/clusters/`*`clusterName`*`/`*`relativeJndiName`*

  where *clusterName* is the name of the server cluster where the object is bound, and *relativeJndiName* is the unqualified name of the object; that is, the object's name relative to a cluster member's server root context.

- **Fixed names**

It is possible to create a fixed name for a server object so that the qualified name is independent of the cell topology. This quality is desirable when clients of the application run in other server processes or as pure clients. Fixed names have the advantage of not changing if the object is moved to another server. The jndiName values in deployment descriptors for a J2EE client application can reference the qualified fixed name for a server object regardless of the cell topology on which the client or server application is being installed.

Defining a cell-wide fixed name for a server application object requires an extra step after the server application is installed. That is, a binding for the object must be created under the cell persistent root. A fixed name bound under the cell persistent root can be any name, but all names under the cell persistent root must be unique within the cell because the cell persistent root is global to the entire cell.

A qualified fixed name has the form:

`cell/persistent/`*`fixedName`*

where *fixedName* is an arbitrary fixed name.

The binding can be created programmatically (for example, using JNDI). However, it is probably more convenient to configure a cell-scoped binding for the server object.

You must keep the programmatic or configured binding up-to-date. Configured EJB bindings are based on the location of the enterprise bean within the cell topology, and moving the EJB application to another single server or to a server cluster, for example, requires the configured binding to be updated. Similar changes affect an EJB home reference programmatically bound so that the fixed name would need to be rebound with a current reference. However, for J2EE clients, the jndiName value for the object, and for thin clients, the lookup name for the object, remains the same. In other words, clients that access objects by fixed names are not affected by changes to the configuration of server applications they access.

JNDI support in WebSphere Application Server

IBM WebSphere Application Server includes a name server to provide shared access to Java components, and an implementation of the `javax.naming` JNDI package which supports user access to the WebSphere Application Server name server through the JNDI naming interface.

WebSphere Application Server does ***not*** provide implementations for:
- `javax.naming.directory` or
- `javax.naming.ldap` packages

Also, WebSphere Application Server does ***not*** support interfaces defined in the `javax.naming.event` package.

However, to provide access to LDAP servers, the development kit shipped with WebSphere Application Server supports the Sun Microsystems implementation of:
- `javax.naming.ldap` and
- `com.sun.jndi.ldap.LdapCtxFactory`

WebSphere Application Server's JNDI implementation is based on version 1.2 of the JNDI interface, and was tested with Version 1.2.1 of the Sun Microsystems JNDI Service Provider Interface (SPI).

The default behavior of this JNDI implementation is adequate for most users. However, users with specific requirements can control certain aspects of JNDI behavior.

Configured name bindings

Administrators can configure bindings into the name space. A configured binding is different from a programmatic binding in that the system creates the binding every time a server is started, even if the target context is in a transient partition.

Administrators can add name bindings to the name space through the configuration. Name servers add these configured bindings to the name space view, by reading the configuration data for the bindings. Configuring bindings is an alternative to creating the bindings from a program. Configured bindings have the advantage of being created each time a server starts, even when the binding is created in a transient partition of the name space. Cell-scoped configured bindings provide a fixed qualified name for server application objects.

Scope

You can configure a binding at one of the following four scopes: cell, node, server, or cluster. Cell-scoped bindings are created under the cell persistent root context. Node-scoped bindings are created under the node persistent root context for the specified node. Server-scoped bindings are created under the server root context for the selected server. Cluster-scoped bindings are created under the server root context in each member of the selected cluster.

The scope you select for new bindings depends on how the binding is to be used. For example, if the binding is not specific to any particular node, cluster, or server, or if you do not want the binding to be associated with any specific node, cluster, or server, a cell-scoped binding is a suitable scope. Defining fixed names for enterprise beans to create fixed qualified names is just such an application. If a binding is to be used only by clients of an application running on a particular server (or cluster), or if you want to configure a binding with the same name on different servers (or clusters) which resolve to different objects, a server-scoped (or cluster-scoped) binding would be appropriate. Note that two servers or clusters can have configured bindings with the same name but resolve to different objects. At the cell scope, only one binding with a given name can exist.

Intermediate contexts

Intermediate contexts created with configured bindings are read-only. For example, if an EJB home binding is configured with the name `some/compound/name/ejbHome`, the intermediate contexts `some`, `some/compound`, and `some/compound/name` will be created as read-only contexts. You cannot add, update, or remove any read-only bindings.

The configured binding name cannot conflict with existing bindings. However, configured bindings can use the same intermediate context names. Therefore, a configured binding with the name `some/compound/name2/ejbHome2` does not conflict with the previous example name.

Configured binding types

Types of objects that you can bind follow:

EJB: EJB home installed in some server in the cell

The following data is required to configure an EJB home binding:

- JNDI name of the EJB server or server cluster where the enterprise bean is deployed
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

A cell-scoped EJB binding is useful for creating a fixed lookup name for an enterprise bean so that the qualified name is not dependent on the topology.

Note: In standalone servers, an EJB binding resolving to another server cannot be configured because the name server does not read configuration data for other servers. That data is required to construct the binding.

CORBA: CORBA object available from some CosNaming name server

You can identify any CORBA object bound into some INS compliant CosNaming server with a corbaname URL. The referenced object does not have to be available until the binding is actually referenced by some application.

The following data is required in order to configure a CORBA object binding:
- The corbaname URL of the CORBA object
- An indicator if the bound object is a context or leaf node object (to set the correct CORBA binding type of context or object)
- Target root for the configured binding
- The name of the configured binding, relative to the target root

Indirect: Any object bound in WebSphere Application Server name space accessible with JNDI
Besides CORBA objects, this includes javax.naming.Referenceable, javax.naming.Reference, and java.io.Serializable objects. The target object itself is not bound to the name space. Only the information required to look up the object is bound. Therefore, the referenced name server does not have to be running until the binding is actually referenced by some application. The following data is required in order to configure an indirect JNDI lookup binding:
- JNDI provider URL of name server where object resides
- JNDI lookup name of object
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

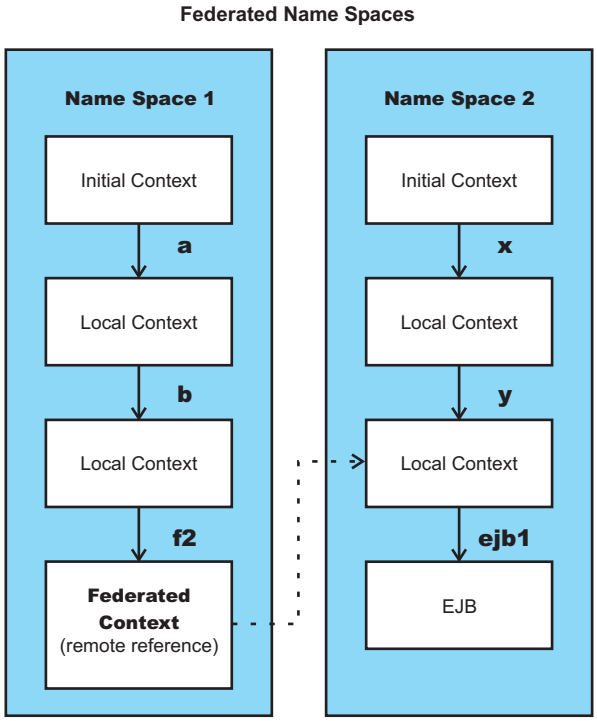
A cell-scoped indirect binding is useful when creating a fixed lookup name for a resource so that the qualified name is not dependent on the topology. You can also achieve this topology by widening the scope of the resource definition.

String: String constant
You can configure a binding of a string constant. The following data is required to configure a string constant binding:
- String constant value
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root

Name space federation

Federating name spaces involves binding contexts from one name space into another name space.

For example, assume that a name space, Name Space 1, contains a context under the name `a/b`. Also assume that a second name space, Name Space 2, contains a context under the name `x/y`. (See the following illustration.) If context `x/y` in Name Space 2 is bound into context `a/b` in Name Space 1 under the name `f2`, the two name spaces are federated. Binding `f2` is a federated binding because the context associated with that binding comes from another name space. From Name Space 1, a lookup of the name `a/b/f2` returns the context bound under the name `x/y` in Name Space 2. Furthermore, if context `x/y` contains an enterprise bean (EJB) home bound under the name `ejb1`, the EJB home can be looked up from Name Space 1 with the lookup name `a/b/f2/ejb1`. Notice that the name crosses name spaces. This fact is transparent to the naming client.



In a WebSphere Application Server name space, you can create federated bindings with the following restrictions:

- Federation is limited to CosNaming name servers. A WebSphere Application Server name server is a Common Object Request Broker Architecture (CORBA) CosNaming implementation. You can create federated bindings to other CosNaming contexts. You cannot, for example, bind contexts from an LDAP name server implementation.
- If you use JNDI to federate the name space, you must use a WebSphere Application Server initial context factory to obtain the reference to the federated context. If you use some other initial context factory implementation, you might not be able to create the binding or the level of transparency might be reduced.
- A federated binding to a non-WebSphere Application Server naming context has the following functional limitations:
  - JNDI operations are restricted to the use of CORBA objects. For example, you can look up EJB homes, but you cannot look up non-CORBA objects such as data sources.
  - JNDI caching is not supported for non-WebSphere Application Server name spaces. This restriction affects the performance of lookup operations only.
  - If security is enabled, WebSphere Application Server does not support federated bindings to non-WebSphereApplication Server name spaces.
- Do not federate two WebSphere Application Server stand-alone server name spaces. Incorrect behavior might result. If you want to federate WebSphere Application Server name spaces, use servers running under the Network Deployment package of WebSphere Application Server.
- When federating the name spaces of two cells running a Network Deployment package of WebSphere Application Server, the names of the cells must be different. Otherwise, incorrect behavior can result.

Naming roles

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization concept is extended to protect the CosNaming service.

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the name space. Generally two ways are acceptable in which client programs result in CosNaming calls. The first is through the Java Naming and Directory Interface (JNDI) methods. The second is CORBA clients invoking CosNaming methods directly.

The following security roles exist. However, the roles have an authority level from low to high as shown in the following list. The list also provides the security-related interface methods for each role. The interface methods that are not listed are either not supported or not relevant to security.

- **CosNamingRead**. Users who are assigned the CosNamingRead role can do queries of the name space, such as through the JNDI lookup method. The Everyone special-subject is the default policy for this role.

Table 44. CosNamingRead role packages and interface methods

Package	Interface methods
javax.naming	• Context.list • Context.listBindings • Context.lookup • NamingEnumeration.hasMore • NamingEnumeration.next
org.omg.CosNaming	• NamingContext.list • NamingContext.resolve • BindingIterator.next_one • BindingIterator.next_n • BindingIterator.destroy

- **CosNamingWrite**. Users who are assigned the CosNamingWrite role can do write operations (such as JNDI bind, rebind, or unbind) plus CosNamingRead operations. As a default policy, Subjects are not assigned this role.

Table 45. CosNamingWrite role packages and interface methods

Package	Interface methods
javax.naming	• Context.bind • Context.rebind • Context.rename • Context.unbind
org.omg.CosNaming	• NamingContext.bind • NamingContext.bind_context • NamingContext.rebind • NamingContext.rebind_context • NamingContext.unbind

- **CosNamingCreate**. Users who are assigned the CosNamingCreate role are allowed to create new objects in the name space through JNDI createSubcontext operations plus CosNamingWrite operations. As a default policy, Subjects are not assigned this role.

Table 46. CosNamingCreate role packages, interface methods

Package	Interface methods
javax.naming	Context.createSubcontext
org.omg.CosNaming	NamingContext.bind_new_context

- **CosNamingDelete**. Users who are assigned the CosNamingDelete role can destroy objects in the name space, for example byusing the JNDI destroySubcontext method and CosNamingCreate operations. As a default policy, Subjects are not assigned this role.

Table 47. CosNamingDelete role packages and interface methods

Package	Interface methods
javax.naming	Context.destroySubcontext
org.omg.CosNaming	NamingContext.destroy

Important: The `javax.naming` package applies to the CosNaming JNDI service provider only. All of the variants of a JNDI interface method have the same role mapping.

If the caller is not authorized, the packages listed in the previous tables exhibit the following behavior:

javax.naming
This package creates the javax.naming.NoPermissionException exception, which maps NO_PERMISSION from the CosNaming method invocation to NoPermissionException.

org.omg.CosNaming
This package creates the org.omg.CORBA.NO_PERMISSION exception.

Users, groups, or the AllAuthenticated and Everyone special subjects can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, you must restart the server for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

If a user is assigned a particular naming role and that user is a member of a group that is assigned a different naming role, the user is granted the most permissive access between the role that is assigned and the role the group is assigned. For example, assume that the MyUser user is assigned the CosNamingRead role. Also, assume that the MyGroup group is assigned the CosNamingCreate role. If the MyUser user is a member of the MyGroup group, the MyUser user is assigned the CosNamingCreate role because the user is a member of the MyGroup group. If the MyUser user is not a member of the MyGroup group, is assigned the CosNamingRead role.

The CosNaming authorization policy is only enforced when administrative security is enabled. When administrative security is enabled, attempts to do CosNaming operations without the proper role assignment result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming server.

In WebSphere Application Server, each CosNaming function is assigned to one role only. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they also are assigned the CosNamingRead role. In most cases, a creator needs three roles assigned: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator example in above have been included in CosNamingCreate role. In most cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, doing so might result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime. Typically, J2EE applications access the name space and the identity is that of the user that authenticated to WebSphere Application Server when he J2EE application is accessed. Unless the J2EE application provider clearly communicates the expected naming roles, fully consider changing the default naming authorization policy.

Naming and directories: Resources for learning

Additional information and guidance on naming and directories is available on various Internet sites.

Use the following links to find relevant supplemental information about naming and directories. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

The naming service provided with WebSphere Application Server Version 6 is the same as that provided for Version 5, thus information on the Version 5 naming and directories applies to Version 6.

The following links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Refer to Web resources for learning for links to information applicable to WebSphere Application Server generally, such as lists of IBM technical papers, Redbooks and samples.

Programming instructions and examples

- Naming in WebSphere Application Server V5: Impact on Migration and Interoperability
- WebSphere Application Server V6 System Management & Configuration Handbook
- *IBM WebSphere Developer Technical Journal:* Co-hosting multiple versions of J2EE applications

Programming specifications

- Java Naming and Directory Interface™ 1.2.1 Specification
- Object Management Group (OMG) Interoperable Naming specifications
  - Naming Service Specification
  - Common Object Request Broker: Architecture and Specification
  - Interoperable Naming Service revised chapters, which presents a consolidated view of all of the elements that comprise interoperable naming

Configuring name servers

Name servers add configured name space bindings to the name space view by reading the configuration data for the bindings. If you use configured bindings, you do not need to create name space bindings from a program.

You can configure a name server to provide a display name and initial state for an application server, as well as specify custom properties for the name server. To configure a name server, complete the following:

1. In the administrative console, click **Servers > Application Servers > Server Components > Name Server**.
2. Edit the fields as desired.

   All of the fields are mandatory.
3. To make other changes, click **Custom Properties** and configure a custom property.
4. Click **OK** to register your changes.

Name server settings

Use this page to configure Naming Service Provider settings for the application server.

To view this administrative console page, click one of the following paths:

- **Servers > Application Servers > *server_name* > Administration > Server Components > Name Server**
- **Servers > JMS Servers > *server_name* > Administration > Server Components > Name Server**

Name
Specifies the display name for the server.

Data type String

Initial State
Specifies the execution state. The options are: *Started* and *Stopped*.

Data type String
Default Started

Configuring name space bindings

Instead of creating name space bindings from a program, you can configure name space bindings using the administrative console. Name servers add these configured bindings to the name space view by reading the configuration data for the bindings. Configured bindings are created each time a server starts, even when the binding is created in a transient partition of the name space. One major use of configured bindings is to provide fixed qualified names for server application objects.

Assemble and deploy your application onto an application server. If the application is a client to an application running in another server process, specify qualified jndiName values for the other application's server objects during assembly or deployment. For more information on qualified names, refer to "Lookup names support in deployment descriptors and thin clients" on page 1775.

A deployed application requires qualified fixed names if the application is accessed by thin client applications or by J2EE client applications or server applications running in another server process.

When you configure a name space binding, you create a qualified fixed name for a server object. A fixed name does not change if the object is moved to another server. A qualified fixed name with a cell scope has the form:

`cell/persistent/`*`fixedName`*

where *fixedName* is an arbitrary fixed name.

You can configure name space bindings, and thus qualified fixed names, for the following objects:
- A string constant value
- An enterprise bean (EJB) home installed on a server in the cell
- A CORBA object available from a CosNaming name server
- An object bound in a WebSphere Application Server name space that is accessible using a JNDI indirect lookup

To view or configure a name space binding for an object of a deployed application, complete the following:

1. Go to the Name Space Bindings page.

   In the administrative console, click **Environment > Naming > Name Space Bindings**.
2. Select the desired scope.

   The scope determines where in the name space the name space binding is created. It also affects which name servers contain the binding in the name space that they manage. Regardless of the scope, a name space binding is accessible from all name servers in the cell. However, the scope can affect whether the lookup can be resolved locally by a name server or whether the name server must make a remote call to another name server to resolve the binding.

   Only name space bindings created with the selected scope are visible in the collection table on the page. By changing the scope, you can see and create bindings in other scopes.

   a. Select a cell, node or server scope.

If you are creating a new name space binding, refer to the table below as a guide in selecting a scope:

Scope	Description
Cell	Cell-scoped bindings are created under the cell persistent root context. Select **Cell** if the name space binding is not specific to any particular node or server, or if you do not want the binding to be associated with any specific node or server. For example, you can use cell-scoped bindings to create fixed qualified names for enterprise beans. Fixed qualified names do not have any node or server names embedded within them. Cell-scoped bindings are created in the deployment manager process, and all node agent and application server processes, in the cell. Therefore all name servers in the cell can resolve those bindings locally. No remote invocations to other name servers are necessary to resolve the bindings.
Node	Node-scoped bindings are created under the node persistent root context for the selected node. Select **Node** if the name space binding is specific to a particular node, or if you want the binding to be associated with a specific node. Node-scoped bindings are created in the node agent and all application server processes in the selected node. Therefore, all name servers in the node can resolve those bindings locally. No remote invocations to other name servers are necessary to resolve the bindings. However, name servers in other nodes must make remote calls to the node agent in the selected node in order to resolve the bindings. For example, in order for a name server running in node *node1* to resolve the name `cell/nodes/node2/persistent/nodeScopedConfiguredBinding`, the name server must make a remote call to the node agent running in *node2*. Any name server running in *node2* can resolve that name without invoking any other name servers.
Server	Server-scoped bindings are created under the server root context for the selected server. Select **Server** if a binding is to be used only by clients of an application running on a particular server, or if you want to configure a binding with the same name on different servers which resolve to different objects. Note that two servers can have configured bindings with the same name but resolve to different objects. Server-scoped bindings are created in the process of the selected application server. Therefore, the name server running in the selected application server can resolve those bindings locally. No remote invocations to other name servers are necessary to resolve the bindings. However, all other name servers in the cell must make remote calls to the selected server in order to resolve the bindings. For example, in order for the name server running in *server1* in node *node1* to resolve the name `cell/nodes/node1/servers/server2/serverScopedConfiguredBinding`, it must make a remote call to *server2* in *node1*. Only the name server in *server2* in *node1* can resolve that name without invoking any other name servers.

b. Click **Apply**.

3. Create a new name space binding.
   a. Open the New Name Space Binding wizard.

      On the Name Space Bindings page, click **New**.
   b. On the **Specify binding type** panel, select the binding type.

      The name space binding can be for a constant string value, an EJB home, a CORBA CosNaming NamingContext or CORBA leaf node object, or an object that you can look up indirectly using JNDI.
   c. On the **Specify basic properties** panel, specify the binding identifier and other properties for the binding.

      For property descriptions, refer to the following:
      - “String binding settings” on page 1787
      - “EJB binding settings” on page 1788
      - “CORBA object binding settings” on page 1789
      - “Indirect lookup binding settings” on page 1790
   d. On the **Summary** panel, verify the settings and click **Finish**.

   The name of the new binding is displayed in the collection table on the Name Space Bindings page.

4. **Optional:** Edit a previously created binding.
   a. From the collection table on the Name Space Bindings page, click the name of the binding that you want to edit.
   b. Edit the binding properties as desired. Step 3(c) provides links to property descriptions.
   c. Click **OK**.

Cell-scoped bindings are created under the cell persistent root context. Node-scoped bindings are created under the node persistent root context for the specified node. Server-scoped bindings are created under the server root context for the selected server. Cluster-scoped bindings are created under the server root context for each member of the selected cluster.

Name space binding collection

Use this page to configure a name binding of an EJB, a CORBA CosNaming NamingContext, a CORBA leaf node object, an object that you can look up using JNDI, or a constant string value.

Binding information for configured bindings is stored in the configuration and applied upon startup of the name server for each server within the scope of the binding.

To view the Name Space Bindings page, click **Environment > Naming > Name Space Bindings**.

Click the check boxes to select one or more of the bindings in your collection. Use the buttons to control the selected bindings.

When creating a new binding, select a scope before clicking **New**. The **Scope** setting filters what bindings are listed in the collection as well as sets the scope of the new binding.

Name

Shows the names given to uniquely identify these configured bindings.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file. This field is for information purposes only and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type

Shows the type of binding configured. Valid values are String, EJB, CORBA, and Indirect. This field is for information purposes only and cannot be updated.

Specify binding type settings

Use this panel to select the type of name space binding that you want.

To view this administrative console panel, click **Environment** > **Naming** > **Name Space Bindings** > **New**.

You can configure a name space binding for any of the following objects:

- A string constant value
- An enterprise bean (EJB) home installed on a server in the cell
- A CORBA object available from a CosNaming name server
- An object bound in a WebSphere Application Server name space that is accessible using a JNDI indirect lookup

On this panel, select a binding type and then click **Next**.

Binding type

Specifies the type of binding configured.

String	Select **String** to configure a name space binding for a string constant value. To configure a String binding, you need the following information: • The string constant value • The target root context for the configured binding (scope) • The name of the configured binding, relative to the target root context
EJB	Select **EJB** to configure a name space binding for an EJB home installed on a server in the cell. Use a cell-scoped EJB binding to create a fixed qualified lookup name for an enterprise bean. A fixed qualified lookup name is not dependent on the cell topology. To configure an EJB home binding, you need the following information: • The JNDI name of the EJB server or server cluster where the enterprise bean is deployed • The target root context for the configured binding (scope) • The name of the configured binding, relative to the target root context On standalone servers, do not configure an EJB binding that resolves to another server. The name server cannot read configuration data for other servers. That data is required to construct the binding.
CORBA	Select **CORBA** to configure a name space binding for a Common Object Request Broker: Architecture and Specification (CORBA) object available from a Object Management Group (OMG) Interoperable Naming (CosNaming) name server. Identify a CORBA object bound into an INS compliant CosNaming server with a corbaname URL. The referenced object does not have to be available until the binding is actually referenced by an application. To configure a CORBA binding, you need the following information: • The corbaname URL of the CORBA object • An indicator if the bound object is a context or leaf node object (to set the correct CORBA binding type of context or object) • The target root context for the configured binding • The name of the configured binding, relative to the target root context
Indirect	Select **Indirect** to configure a name space binding for an object bound in a WebSphere Application Server name space that is accessible using a Java Naming and Directory Interface (JNDI) indirect lookup. You can select **Indirect** for CORBA objects as well as for javax.naming.Referenceable, javax.naming.Reference, and java.io.Serializable objects. The target object itself is not bound to the name space. Only the information required to look up the object is bound. Therefore, the referenced name server does not have to be running until the binding is actually referenced by some application. To configure an indirect JNDI lookup binding, you need the following information: • The JNDI provider URL of the name server where the object resides • The JNDI lookup name of the object • The target root context for the configured binding (scope) • The name of the configured binding, relative to the target root context The following information is optional: • The JNDI initial context factory class name. The default is the WebSphere Application Server initial context factory, com.ibm.websphere.naming.WsnInitialContextFactory. • Additional properties to pass to the javax.naming.InitialContext constructor. A cell-scoped indirect binding is useful when creating a fixed qualified lookup name for a bound object so that the qualified lookup name is not dependent on the cell topology.

String binding settings

Use this page to view or configure a string binding.

To view this console page, click **Environment > Naming > Name Space Bindings >** *string_namespace_binding*. The settings on this page are similar to those on the **Specify basic properties** panel of the New Name Space Binding wizard.

Scope
Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file.

The **Scope** setting is shown only when you edit an existing binding on the console page, and is not shown when you create a new binding on the wizard panel. This setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type
Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes only and cannot be updated.

Binding identifier
Specifies the name that uniquely identifies this configured binding.

Name in name space
Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

String value
Specifies the string to be bound into the name space.

EJB binding settings
Use this page to configure a new EJB binding, or to view or edit an existing EJB binding.

To view this console page, click **Environment > Naming > Name Space Bindings >** *EJB_namespace_binding*. The settings on this page are similar to those on the **Specify basic properties** panel of the New Name Space Binding wizard.

Scope
Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file.

The **Scope** setting is shown only when you edit an existing binding on the console page, and is not shown when you create a new binding on the wizard panel. This setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type
Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes only and cannot be updated.

Binding identifier
Specifies the name that uniquely identifies this configured binding.

Name in name space
Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

Enterprise Bean Location
Specifies whether the enterprise bean is running in a server cluster or a single server. If Single server is specified, type the node name.

Server
Specifies the name of the cluster or non-clustered server in which the enterprise bean is configured.

JNDI name
Specifies the JNDI name of the deployed enterprise bean (the bean's JNDI name that is in the enterprise bean bindings--not the java:comp name).

CORBA object binding settings
Use this page to configure a new name binding of a CORBA object binding, or to view or edit an existing CORBA object binding.

To view this console page, click **Environment > Naming > Name Space Bindings >** *CORBA_namespace_binding*. The settings on this page are similar to those on the **Specify basic properties** panel of the New Name Space Binding wizard.

Scope
Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file.

The **Scope** setting is shown only when you edit an existing binding on the console page, and is not shown when you create a new binding on the wizard panel. This setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type
Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes only and cannot be updated.

Binding identifier
Specifies the name that uniquely identifies this configured binding.

Name in name space
Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

Corbaname URL
Specifies the CORBA name URL string identifying where the object is bound in a CosNaming server.

Federated context
Specifies whether the target is a CosNaming context (true) or a leaf node object (false).

true — The target object is bound with a context CORBA binding type. If the corbaname URL does not resolve to a NamingContext, an error occurs when the binding is first used (which is when the URL is first resolved).

false	The target object is bound with an object CORBA binding type.

Indirect lookup binding settings

Use this page to configure a new indirect lookup name binding, or to view or edit an existing indirect lookup binding.

To view this console page, click **Environment > Naming > Name Space Bindings >** *indirect_lookup_namespace_binding*. The settings on this page are similar to those on the **Specify basic properties** panel of the New Name Space Binding wizard.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the `namebindings.xml` file.

The **Scope** setting is shown only when you edit an existing binding on the console page, and is not shown when you create a new binding on the wizard panel. This setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes only and cannot be updated.

Binding identifier

Specifies the name that uniquely identifies this configured binding.

Name in name space

Specifies the name used for this binding in the name space. This name can be a simple or compound name depending on the portion of the name space where this binding is configured.

Provider URL

Specifies the provider URL string needed to obtain a JNDI initial context.

JNDI name

Specifies the name used to look up the target object from the initial context.

Initial context factory name

Specifies the class name of the initial context factory used to obtain a JNDI initial context.

This field is optional. If no factory is specified, the WebSphere Application Server initial context factory is used.

If the scope of the indirect lookup binding includes servers or nodes previous to version 6.1, the initial context factory name and other context factory properties are not shown on the console page.

Developing applications that use JNDI

References to EJB homes and other artifacts such as data sources are bound to the WebSphere Application Server name space. These objects can be obtained through the JNDI interface. Before you can perform any JNDI operations, you need to get an initial context. You can use the initial context to look up objects bound to the WebSphere Application Server name space.

The following examples describe how to get an initial context and how to perform lookup operations.
- Getting the default initial context
- Getting an initial context by setting the provider URL property
- Setting the provider URL property to select a different root context as the initial context
- Looking up an EJB home with JNDI
- Looking up a JavaMail session with JNDI

In these examples, the default behavior of features specific to the WebSphere Application Server JNDI Context implementation is used.

The WebSphere Application Server JNDI context implementation includes special features. JNDI caching enhances performance of repeated lookup operations on the same objects. Name syntax options offer a choice of a name syntaxes, one optimized for typical JNDI clients, and one optimized for interoperability with CosNaming applications. Most of the time, the default behavior of these features is the preferred behavior. However, sometimes you should modify the behavior for specific situations.

JNDI caching and name syntax options are associated with a javax.naming.InitialContext instance. To select options for these features, set properties that are recognized by the WebSphere Application Server initial context factory. To set JNDI caching or name syntax properties which will be visible to WebSphere Application Server initial context factory, do the following:

1. **Optional:** Configure JNDI caches

   JNDI caching can greatly increase performance of JNDI lookup operations. By default, JNDI caching is enabled. In most situations, this default is the desired behavior. However, in specific situations, use the other JNDI cache options.

   Objects are cached locally as they are looked up. Subsequent lookups on cached objects are resolved locally. However, cache contents can become stale. This situation is not usually a problem, since most objects you look up do not change frequently. If you need to look up objects which change relatively frequently, change your JNDI cache options.

   JNDI clients can use several properties to control cache behavior.

   You can set properties:

   - From the command line by entering the actual string value. For example:

     ```
     java -Dcom.ibm.websphere.naming.jndicache.maxentrylife=1440
     ```

   - In a `jndi.properties` file by creating a file named `jndi.properties` as a text file with the desired properties settings. For example:

     ```
     ...
     com.ibm.websphere.naming.jndicache.cacheobject=none
     ...
     ```

     Include the file as the beginning of the classpath, so that the class loader loads your copy of `jndi.properties` before any other copies.

   - Within a Java program by using the **PROPS.JNDI_CACHE*** Java constants, defined in the ***com.ibm.websphere.naming.PROPS*** file. The constant definitions follow:

     ```
     public static final String JNDI_CACHE_OBJECT =
      "com.ibm.websphere.naming.jndicache.cacheobject";
     public static final String JNDI_CACHE_OBJECT_NONE      = "none";
     public static final String JNDI_CACHE_OBJECT_POPULATED = "populated";
     public static final String JNDI_CACHE_OBJECT_CLEARED   = "cleared";
     public static final String JNDI_CACHE_OBJECT_DEFAULT   =
      JNDI_CACHE_OBJECT_POPULATED;

     public static final String JNDI_CACHE_NAME =
      "com.ibm.websphere.naming.jndicache.cachename";
     public static final String JNDI_CACHE_NAME_DEFAULT = "providerURL";

     public static final String JNDI_CACHE_MAX_LIFE =
      "com.ibm.websphere.naming.jndicache.maxcachelife";
     public static final int    JNDI_CACHE_MAX_LIFE_DEFAULT = 0;
     ```

```
public static final String JNDI_CACHE_MAX_ENTRY_LIFE =
 "com.ibm.websphere.naming.jndicache.maxentrylife";
public static final int    JNDI_CACHE_MAX_ENTRY_LIFE_DEFAULT = 0;
```

To use the previous properties in a Java program, add the property setting to a hashtable and pass it to the InitialContext constructor as follows:

```
java.util.Hashtable env = new java.util.Hashtable();
...

// Disable caching
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE); ...
javax.naming.Context initialContext = new javax.naming.InitialContext(env);
```

2. **Optional:** Specify the name syntax

   Most WebSphere applications use JNDI to look up EJB objects and do not need to look up objects bound by CORBA applications. Therefore, the default name syntax used for JNDI names is the most convenient. If your application needs to look up objects bound by CORBA applications, you may need to change your name syntax so that all CORBA CosNaming names can be represented.

   JNDI clients can set the name syntax by setting a property. The property setting is applied by the initial context factory when you instantiate a new java.naming.InitialContext object. Names specified in JNDI operations on the initial context are parsed according to the specified name syntax.

   You can set the property:

   - From the command line by entering the actual string value. For example:

     ```
     java -Dcom.ibm.websphere.naming.name.syntax=ins
     ```

   - In a `jndi.properties` file by creating a file named `jndi.properties` as a text file with the desired properties settings. For example:

     ```
     ...
     com.ibm.websphere.naming.name.syntax=ins
     ...
     ```

     Include the file at the beginning of the classpath, so that the class loader loads your copy of `jndi.properties` before any other copies.

   - Within a Java program by using the PROPS.NAME_SYNTAX* Java constants, defined in the `com.ibm.websphere.naming.PROPS` file. The constant definitions follow:

     ```
     public static final String NAME_SYNTAX =
         "com.ibm.websphere.naming.name.syntax";
     public static final String NAME_SYNTAX_JNDI = "jndi";
     public static final String NAME_SYNTAX_INS  = "ins";
     ```

     To use the previous properties in a Java program, add the property setting to a hashtable and pass it to the InitialContext constructor as follows:

     ```
     java.util.Hashtable env = new java.util.Hashtable();
     ...
     env.put(PROPS.NAME_SYNTAX, PROPS.NAME_SYNTAX_INS); // Set name syntax to INS
     ...
     javax.naming.Context initialContext = new javax.naming.InitialContext(env);
     ```

Example: Getting the default initial context

There are various ways for a program to get the default initial context.

The following example gets the default initial context. Note that no provider URL is passed to the javax.naming.InitialContext constructor.

```
...
import javax.naming.Context;
import javax.naming.InitialContext;
...
Context initialContext = new InitialContext();
...
```

The default initial context returned depends the runtime environment of the JNDI client. Following are the initial contexts returned in the various environments:

Thin client
The initial context is the server root context of the server running on the local host at port 2809.

Pure client
The initial context is the context specified by the java.naming.provider.url property passed to launchClient command with the -CCD command line parameter. The context usually is the server root context of the server at the address specified in the URL, although it is possible to construct a corbaname or corbaloc URL which resolves to some other context.

If no provider URL is specified, it is the server root context of the server running on the host and port specified by the `-CCproviderURL`, or `-CCBootstrapHost` and `-CCBootstrapPort` command line parameters. The default host is the local host, and the default port is 2809.

Server process
The initial context is the server root context for that process.

Even though no provider URL is explicitly specified in the above example, the InitialContext constructor might find a provider URL defined in other places that it searches for property settings.

Users of properties which affect ORB initialization should read the rest of this section for a deeper understanding of exactly how initial contexts are obtained.

Determining which server is used to obtain the initial context

WebSphere Application Server name servers are CORBA CosNaming name servers, and WebSphere Application Server provides a CosNaming JNDI plug-in implementation for JNDI clients to perform naming operations on WebSphere Application Server name spaces. The WebSphere Application Server CosNaming plug-in implementation is selected through a JNDI property that is passed to the InitialContext constructor. This property is `java.naming.factory.initial`, and it specifies the initial context factory implementation to use to obtain an initial context. The factory returns a javax.naming.Context instance, which is part of its implementation.

The WebSphere Application Server initial context factory, `com.ibm.websphere.naming.WsnInitialContextFactory`, is typically used by WebSphere Application Server applications to perform JNDI operations. The WebSphere Application Server runtime environment is set up to use this WebSphere Application Server initial context factory if one is not specified explicitly by the JNDI client. When the initial context factory is invoked, an *initial context* is obtained. The following paragraphs explain how the WebSphere Application Server initial context factory obtains the initial context in client and server environments.

- **Registration of initial references in server processes**

  Every WebSphere Application Server has an ORB used to receive and dispatch invocations on objects running in that server. Services running in the server process can register initial references with the ORB. Each initial reference is registered under a key, which is a string value. An initial reference can be any CORBA object. WebSphere Application Server name servers register several initial contexts as initial references under predefined keys. Each name server initial reference is an instance of the interface org.omg.CosNaming.NamingContext.

- **Obtaining initial references in pure client processes**

  Pure JNDI clients, that is, JNDI clients which are not running in a WebSphere Application Server process, also have an ORB instance. This client ORB instance can be passed to the InitialContext constructor, but typically the initial context factory creates and initializes the client ORB instance transparently. A client ORB can be initialized with initial references, but the initial references most likely resolve to objects running in some server. The initial context factory does not define any default initial references when it initializes an ORB. If the `resolve_initial_references` method is invoked on the client ORB when no initial references have been configured, the method invocation fails. This condition

is typical for pure client processes. To obtain an initial NamingContext reference, the initial context factory must invoke `string_to_object` with an IIOP type CORBA object URL, such as `corbaloc:iiop:myhost:2809`. The URL specifies the address of the server from which to obtain the initial context. The host and port information is extracted from the provider URL passed to the InitialContext constructor.

If no provider URL is defined, the WebSphere Application Server initial context factory uses the default provider URL of `corbaloc:iiop:localhost:2809`. The `string_to_object` ORB method resolves the URL and communicates with the target server ORB to obtain the initial reference.

- **Obtaining initial references in server processes**

  If the JNDI client is running in a WebSphere Application Server process, the initial context factory obtains a reference to the server ORB instance if the JNDI client does not provide an ORB instance. Typically, JNDI clients running in server processes use the server ORB instance; that is, they do not pass an ORB instance to the InitialContext constructor. The name server which is running in the server process sets a provider URL as a java.lang.System property to serve as the default provider URL for all JNDI clients in the process. This default provider URL is `corbaloc:rir:/NameServiceServerRoot`. This URL resolves to the server root context for that server. (The URL is equivalent to invoking `resolve_initial_references` on the ORB with a key of NameServiceServerRoot. The name server registers the server root context as an initial reference under that key.)

- **Understanding the legacy ORB protocol**

  Releases previous to WebSphere Application Server Version 5 used a different ORB implementation, which used a legacy protocol in contrast with the Interoperable Name Service (INS) protocol now used. This change has affected the implementation of the WebSphere Application Server initial context factory. **Certain types of pure clients can experience different behavior when getting initial JNDI contexts as compared to previous releases of WebSphere Application Server**. This behavior is discussed in more detail below.

  The following ORB properties are used with the legacy ORB protocol for ORB initialization and are now deprecated:
  – com.ibm.CORBA.BootstrapHost
  – com.ibm.CORBA.BootstrapPort

  The new INS ORB is different in a major respect, in that it exhibits no default behavior if no initial references are defined.

  In the legacy ORB, the bootstrap host and port values defaulted to `localhost` and `900`.

  All initial references were obtained from the server running on the bootstrap host and port. So, if the ORB user provided no bootstrap host and port, all initial references are resolved from the server running on the local host at port 900. The INS ORB has no concept of bootstrap host or bootstrap port. All initial references are defined independently. That is, different initial references could resolve to different servers. If `ORB.resolve_initial_references` is invoked with a key such that the ORB is not initialized with an initial reference having that key, the call fails.

  In releases of WebSphere Application Server previous to Version 5, the initial context factory invoked `resolve_initial_references` on the ORB in the absence of any provider URL. This action succeeded if a name server at the default bootstrap host and port was running. In the current release, with the INS ORB, this would fail. (Actually, the ORB would fall back to the legacy protocol during the deprecation period, but when the legacy protocol is no longer supported, the operation would fail.)

  The initial context factory now uses a default provider URL of `corbaloc:iiop:localhost:2809`, and invokes `string_to_object` with the provider URL.

  This operation preserves the behavior that pure clients in previous releases experienced when they set no ORB bootstrap properties or provider URL. **However, this different initial context factory implementation changes the behavior experienced by certain legacy pure clients, which do not specify a provider URL**:
  – Clients which set the ORB bootstrap properties listed above when getting an initial context.
  – Clients which supply their own ORB instance to the InitialContext constructor.

  There are two ways to circumvent this change of behavior:

– Always specify an IIOP type provider URL. This approach does not depend on the bootstrap host and port properties and continues to work when support for the bootstrap host and port properties is removed. For example, you can express bootstrap host and port property values of `myHost` and `2809`, respectively, as `corbaloc:iiop:myHost:2809`.
– Use an rir type provider URL:
  - Specify `corbaloc:rir:/NameServiceServerRoot` if the ORB is initialized to use a WebSphere Application Server 5 server as the bootstrap server.
  - Specify `corbaname:rir:/NameService#domain/legacyRoot` if the ORB is initialized to use a WebSphere Application Server 4.0.x server as the bootstrap server.
  - Specify `corbaloc:rir:/NameService` if the ORB is initialized to use a server other than a WebSphere Application Server 5 or 4.0.x server as the bootstrap server.

  URLs of this type are equivalent to invoking `resolve_initial_references` on the ORB with the specified key. If the bootstrap host and port properties are being used to initialize the ORB, this approach will not work when the bootstrap and host properties are no longer supported.

- **The InitialContext constructor search order for JNDI properties**

  If the code snippet shown at the beginning of this section is executed by an application, the bootstrap server depends on the value of the property, java.naming.provider.url. If the property is not set (in server processes the default value is set as a system property), the default host of `localhost` and default port of `2809` are used as the address of the server from which to obtain the initial context. The JNDI specification describes where the InitialContext constructor looks for java.naming.provider.url property settings, but briefly, the property is picked up from the following places in the order shown:

  InitialContext constructor
  This does not apply to the above example because the example uses the empty InitalContext constructor.

  System environment
  You can add JNDI properties to the system environment as an option on the Java command invocation and by program code. The recommended way to set the provider URL in the system environment is as an option supplied to the Java command invocation. Setting the provider URL in this manner is not temporal, so that getting a default initial context will always yield the same result. It is generally recommended that program code not set the provider URL property in the system environment because as a side-effect, this could adversely affect other, possibly unrelated, code running elsewhere in the same process.

  `jndi.properties` file
  There may be many `jndi.properties` files that are within the scope of the class loader in effect. All `jndi.properties` files are used for setting JNDI properties, but the provider URL setting is determined by the first `jndi.properties` file returned by the class loader.

Example: Getting an initial context by setting the provider URL property

In general, JNDI clients should assume the correct environment is already configured so there is no need to explicitly set property values and pass them to the `InitialContext` constructor. However, a JNDI client might need to access a name space other than the one identified in its environment. In this case, it is necessary to explicitly set the java.naming.provider.url (provider URL) property used by the `InitialContext` constructor. A provider URL contains bootstrap server information that the initial context factory can use to obtain an initial context. Any property values passed in directly to the `InitialContext` constructor take precedence over settings of those same properties found elsewhere in the environment.

You can use two different provider URL forms with WebSphere Application Server's initial context factory:
- A CORBA object URL (new for J2EE 1.3)
- An IIOP URL

CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. CORBA object URLs are part of the OMG CosNaming Interoperable Naming Specification. A corbaname

URL, for example, can include initial context and lookup name information and can be used as a lookup name without the need to explicitly obtain another initial context. The IIOP URLs are the legacy JNDI format, but are still supported by the WebSphere Application Server initial context factory.

The following examples illustrate the use of these URLs.

- "Using a CORBA object URL"
- "Using a CORBA object URL with multiple name server addresses"
- "Using a CORBA object URL from a non-WebSphere Application Server JNDI implementation"
- "Using an IIOP URL" on page 1797

Using a CORBA object URL

This example shows a CORBA object URL.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
     "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
...
```

Using a CORBA object URL with multiple name server addresses

CORBA object URLs can contain more than one bootstrap address. You can use this feature when attempting to obtain an initial context from a server cluster. You can specify the bootstrap addresses for all servers in the cluster in the URL. The operation succeeds if at least one of the servers is running, eliminating a single point of failure. There is no guarantee of any particular order in which the address list will be processed. For example, the second bootstrap address may be used to obtain the initial context even though the server at the first bootstrap address in the list is available.

Multiple-address provider URLs resolving to servers on non-z/OS systems cannot contain bootstrap addresses for node agent processes. The URLs should only contain the bootstrap addresses of members of the same cluster. Otherwise, incorrect behavior might occur. When resolving to servers running on the z/OS operating system, the URL can contain bootstrap addresses for node agent processes.

An example of a corbaloc URL with multiple addresses follows.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
     "com.ibm.websphere.naming.WsnInitialContextFactory");
// All of the servers in the provider URL below are members of
// the same cluster.
env.put(Context.PROVIDER_URL,
     "corbaloc::myhost1:9810,:myhost1:9811,:myhost2:9810");
Context initialContext = new InitialContext(env);
...
```

Using a CORBA object URL from a non-WebSphere Application Server JNDI implementation

Initial context factories for CosNaming JNDI plug-in implementations other than the WebSphere Application Server initial context factory most likely obtain an initial context using the object key, `NameService`. When you use such a context factory to obtain an initial context from a WebSphere Application Server name server, the initial context is the cell root context. Since system artifacts such as EJB homes associated

with a server are bound under the server's server root context, names used in JNDI operations must be qualified. If you want to use relative names, ensure your initial context is the server root context under which the target object is bound. In order to make the server root context the initial context, specify a corbaloc provider URL with an object key of `NameServiceServerRoot`.

This example shows a CORBA object type URL from a non-WebSphere Application Server JNDI implementation. This example assumes full CORBA object URL support by the non-WebSphere Application Server JNDI implementation. The object key of `NameServiceServerRoot` is specified so that the initial context will be the specified server's server root context.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.somecompany.naming.TheirInitialContextFactory");
env.put(Context.PROVIDER_URL,
      "corbaname:iiop:myhost.mycompany.com:9810/NameServiceServerRoot");
Context initialContext = new InitialContext(env);
...
```

If qualified names are used, you can use the default key of `NameService`.

Using an IIOP URL

The IIOP type of URL is a legacy format which is not as flexible as CORBA object URLs. However, URLs of this type are still supported. The following example shows an IIOP type URL as the provider URL.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
     "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "iiop://myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
...
```

Example: Setting the provider URL property to select a different root context as the initial context

Each server contains its own server root context, and, when bootstrapping to a server, the server root is the default initial JNDI context. Most of the time, this default is the desired initial context, since system artifacts such as EJB homes are bound there. However, other root contexts exist, which can contain bindings of interest. It is possible to specify a provider URL to select other root contexts.

Examples for selecting other root contexts follow:

- Initial root contexts with a CORBA object URL
- Initial root contexts with the name space root property

Selecting the initial root context with a CORBA object URL

There are several object keys registered with the bootstrap server that you can use to select the root context for the initial context. To select a particular root context with a CORBA object URL object key, set the object key to the corresponding value. The default object key is NameService. Using JNDI yields the server root context. A table that lists the different root contexts and their corresponding object key follows:

Root Context	CORBA Object URL Object Key
Server Root	NameServiceServerRoot

Root Context	CORBA Object URL Object Key
Cell Persistent Root	NameServiceCellPersistentRoot
Cell Root	NameServiceCellRoot
Node Root	NameServiceNodeRoot

The following example shows the use of a corbaloc URL with the object key set to select the cell persistent root context as the initial context.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
      "corbaloc:iiop:myhost.mycompany.com:2809/NameServiceCellPersistentRoot");
Context initialContext = new InitialContext(env);
...
```

Selecting the initial root context with the name space root property

You can also select the initial root context by passing a name space root property setting to the InitialContext constructor. Generally, the object key setting described above is sufficient. Sometimes a property setting is preferable. For example, you can set the root context property on the Java invocation to make which server root is being used as the initial context transparent to the application. The default server root property setting is `defaultroot`, which yields the server root context.

Root Context	Name Space Root Property Value
Server Root	bootstrapserverroot
Cell Persistent Root	cellpersistentroot
Cell Root	cellroot
Node Root	bootstrapnoderoot

The initial context factory ignores the name space root property if the provider URL contains an object key other than `NameService`.

The following example shows use of the name space root property to select the cell persistent root context as the initial context. Note that available constants are used instead of hard-coding the property name and value.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
env.put(PROPS.NAME_SPACE_ROOT, PROPS.NAME_SPACE_ROOT_CELL_PERSISTENT);
Context initialContext = new InitialContext(env);
...
```

Example: Looking up an EJB home with JNDI

Most applications that use JNDI run in a container. Some do not. The name used to look up an object depends on whether or not the application is running in a container. Sometimes it is more convenient for an application to use a corbaname URL as the lookup name. Container-based JNDI clients and thin Java clients can use a corbaname URL.

The following examples show how to perform JNDI lookups from different types of applications.
- "JNDI lookup from an application running in a container"
- "JNDI lookup from an application that does not run in a container"
- "JNDI lookup with a corbaname URL" on page 1800

JNDI lookup from an application running in a container

Applications that run in a container can use `java:` lookup names. Lookup names of this form provide a level of indirection such that the lookup name used to look up an object is not dependent on the object's name as it is bound in the name server's name space. The deployment descriptors for the application provide the mapping from the `java:` name and the name server lookup name. The container sets up the `java:` name space based on the deployment descriptor information so that the `java:` name is correctly mapped to the corresponding object.

The following example shows a lookup of an EJB home. The actual home lookup name is determined by the application's deployment descriptors. The enterprise bean (EJB) resides in an EJB container, which provides an interface between the bean and the application server on which it resides.

```
// Get the initial context as shown in a previous example
...
// Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome =
      initialContext.lookup(
         "java:comp/env/com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
   catch (NamingException e) { // Error getting the home interface
   ...
}
```

JNDI lookup from an application that does not run in a container

Applications that do not run in a container cannot use `java:` lookup names because it is the container which sets the `java:` name space up for the application. Instead, an application of this type must look the object up directly from the name server. Each application server contains a name server. System artifacts such as EJB homes are bound relative to the server root context in that name server. The various name servers are federated by means of a system name space structure. The recommended way to look up objects on different servers is to qualify the name so that the name resolves from any initial context in the cell. If a relative name is used, the initial context must be the same server root context as the one under which the object is bound. The form of the qualified name depends on whether the qualified name is a topology-based name or a fixed name. A topology based name depends on whether the object resides in a single server or a server cluster. Examples of each form of qualified name follow.

- **Topology-based qualified names**

  Topology-based qualified names traverse through the system name space to the server root context under which the target object is bound. A topology-based qualified name resolves from any initial context in the cell. The topology-based qualified name depends on whether the object resides on a single server or server cluster. Examples of each lookup follow.

Single server

The following example shows a lookup of an EJB home that is running in the single server, MyServer, configured in the node, Node1.

```
// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome = initialContext.lookup(
      "cell/nodes/Node1/servers/MyServer/com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
catch (NamingException e) { // Error getting the home interface
   ...
}
```

Server cluster

The example below shows a lookup of an EJB home which is running in the cluster, MyCluster. The name can be resolved if any of the cluster members is running.

```
// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
   // Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome = initialContext.lookup(
      "cell/clusters/MyCluster/com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
catch (NamingException e) { // Error getting the home interface
   ...
}
```

- **Fixed qualified names**

  If the target object has a cell-scoped fixed name defined for it, you can use its qualified form instead of the topology-based qualified name. Even though the topology-based name works, the fixed name does not change with the specific cell topology or with the movement of the target object to a different server. An example lookup with a qualified fixed name is shown below.

```
// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome = initialContext.lookup(
     "cell/persistent/com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
   }
catch (NamingException e) { // Error getting the home interface
...
}
```

JNDI lookup with a corbaname URL

A corbaname can be useful at times as a lookup name. If, for example, the target object is not a member of the federated name space and cannot be located with a qualifiied name, a corbaname can be a convenient way to look up the object. A lookup with a corbaname URL follows.

```
// Get the initial context as shown in a previous example
...
// Look up the home interface using a corbaname URL
try {
   java.lang.Object ejbHome = initialContext.lookup(
      "corbaname:iiop:someHost:2809#com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
catch (NamingException e) { // Error getting the home interface
   ...
}
```

Example: Looking up a JavaMail session with JNDI

A program can conduct a JNDI lookup of a JavaMail resource. Deployment descriptors of the application determine the lookup name that you specify.

The following example shows a lookup of a JavaMail resource:

```
// Get the initial context as shown above
...
Session session =
     (Session) initialContext.lookup("java:comp/env/mail/MailSession");
```

JNDI interoperability considerations

You must take extra steps to enable your programs to interoperate with non-WebSphere Application Server JNDI clients and to bind resources from MQSeries to a name space.

EJB clients running in an environment other than WebSphere Application Server accessing EJB applications running on WebSphere Application Server V5 or V6 servers

When an EJB application running in WebSphere Application Server V5 or V6 is accessed by a non-WebSphere Application Server EJB client, the JNDI initial context factory is presumed to be a non-WebSphere Application Server implementation. In this case, the default initial context is the cell root. If the JNDI service provider being used supports CORBA object URLs, the corbaname format can be used to look up the EJB home. The construction of the stringified name depends on whether the object is installed on a single server or cluster.

Single server

Following is a URL that has the bootstrap host **myHost**, the port **2809**, and the enterprise bean installed in the server **server1** in node **node1** and bound in that server under the name **myEJB**:

```
initialContext.lookup(
  "corbaname:iiop:myHost:2809#cell/nodes/node1/servers/server1/myEJB");
```

Server cluster

Following is a URL that has the bootstrap host **myHost**, the port **2809**, and the enterprise bean installed in a server cluster named **myCluster** and bound in that cluster under the name **myEJB**:

```
initialContext.lookup(
    "corbaname:iiop:myHost:2809#cell/clusters/myCluster/myEJB");
```

The lookup works with any name server bootstrap host and port configured in the same cell.

The lookup also works if the bootstrap host and port belong to a member of the cluster itself. To avoid a single point of failure, the bootstrap server host and port for each cluster member can be listed in the URL as follows:

```
initialContext.lookup(
    "corbaname:iiop:host1:9810,:host2:9810#cell/clusters/myCluster/myEJB");
```

The name prefix **cell/clusters/myCluster/** is not necessary if boostrapping to the cluster itself, but it will work. The prefix is needed, however, when looking up enterprise beans in other clusters. Name bindings under the **clusters** context are implemented on the name server to resolve to the server root of a running cluster member during a lookup; thus avoiding a single point of failure.

Without CORBA object URL support

If the JNDI initial context factory being used does not support CORBA object URLs, the initial context can be obtained from the server, and the lookup can be performed on the initial context as follows:

```
Hashtable env = new Hashtable();
env.put(CONTEXT.PROVIDER_URL, "iiop://myHost:2809");
Context ic = new InitialContext(env);
Object o = ic.lookup("cell/clusters/myCluster/myEJB");
```

Binding resources from MQSeries 5.2

In releases previous to WebSphere Application Server V5, the MQSeries jmsadmin tool could be used to bind resources to the name space. When used with a WebSphere Application Server V5 or V6 name space, the resource is bound within a transient partition in the name space and does not persist past the life of the server process. Instead of binding the MQSeries resources with the jmsadmin tool, bind them from the WebSphere Application Server administrative console, under **Resources** in the console navigation tree.

JNDI caching

To increase the performance of JNDI operations, the WebSphere Application Server JNDI implementation employs caching to reduce the number of remote calls to the name server for lookup operations. For most cases, use the default cache setting.

When an InitialContext object is instantiated, an association is established between the InitialContext instance and a cache. The initial context and any contexts returned directly or indirectly from a lookup on the initial context are all associated with that same cache instance. By default, the association is based on the provider URL, in particular, the host name and port. The caller can specify the cache name to override this default behavior. A cache instance of a given name is shared by all instances of InitialContext configured to use a cache of that name which were created with the same context class loader in effect. Two EJB applications running in the same server will use their own cache instances, if they are using different context class loaders, even if the cache names are the same.

After an association between an InitialContext instance and cache is established, the association does not change. A `javax.naming.Context` object returned from a lookup operation inherits the cache association of the Context object on which the lookup was performed. Changing cache property values with the `Context.addToEnvironment()` or `Context.removeFromEnvironment()` method does not affect cache behavior. You can change properties affecting a given cache instance with each InitialContext instantiation.

A cache is restricted to a process and does not persist past the life of that process. A cached object is returned from lookup operations until either the maximum cache life for the cache is reached, or the maximum entry life for the object's cache entry is reached. After this time, a lookup on the object causes the cache entry for the object to be refreshed. By default, caches and cache entries have unlimited lifetimes.

Usually, cached objects are relatively static entities, and objects becoming stale is not a problem. However, you can set timeout values on cache entries or on a cache so that cache contents are periodically refreshed.

If a bind or rebind operation is executed on an object, the change is not reflected in any caches other than the one associated with the context from which the bind or rebind was issued. This scenario is most likely

to happen when multiple processes are involved, since different processes do not share the same cache, and context objects in all threads in a process typically share the same cache instance for a given name service provider.

JNDI cache settings

Various cache property settings follow. Ensure that all property values are string values.

com.ibm.websphere.naming.jndicache.cachename

The name of the cache to associate with an initial context instance can be specified with this property.

It is possible to create multiple InitialContext instances, each operating on the name space of a different name server. By default, objects from each bootstrap address are cached separately, since they each involve independent name spaces and name collisions could occur if they used the same cache. The provider URL specified when the initial context is created by default serves as the basis for the cache name. With this property, a JNDI client can specify a cache name. Valid options for cache names follow:

Valid options	Resulting cache behavior
providerURL (default)	Use the value for java.naming.provider.url property as the basis for the cache name. Cache names are based on the bootstrap host and port specified in the URL. The boostrap host is normalized to a fully qualfied name, if possible. For example, ″corbaname:iiop:server1:2809#some/starting/context″ and ″corbaloc:iiop://server1″ are normalized to the same cache name. If no provider URL is specified, a default cache name is used.
Any string	Use the specified string as the cache name. You can use any arbitrary string with a value other than ″providerURL″ as a cache name.

com.ibm.websphere.naming.jndicache.cacheobject

Turn caching on or off and clear an existing cache with this property.

By default, when an InitialContext is instantiated, it is associated with an existing cache or, if one does not exist, a new one is created. An existing cache is used with its existing contents. In some circumstances, this behavior is not desirable. For example, when objects that are looked up change frequently, they can become stale in the cache. Other options are available. The following table lists these other options along with the corresponding property value.

Valid values	Resulting cache behavior
populated (default)	Use a cache with the specified name. If the cache already exists, leave existing cache entries in the cache; otherwise, create a new cache.
cleared	Use a cache with the specified name. If the cache already exists, clear all cache entries from the cache; otherwise, create a new cache.
none	Do not cache. If this option is specified, the cache name is irrelevant. Therefore, this option will not disable a cache that is already associated with other InitialContext instances. The InitialContext that is instantiated is not associated with any cache.

com.ibm.websphere.naming.jndicache.maxcachelife

Impose a limit to the age of a cache with this property.

By default, cached objects remain in the cache for the life of the process or until cleared with the com.ibm.websphere.naming.jndicache.cacheobject property set to ″cleared″. This property enables a JNDI client to set the maximum life of a cache. This property differs from the maxentrylife property (below) in that the entire cache is cleared when the cache lifetime is reached. The table below lists the various maxcachelife values and their affect on cache behavior:

Valid options	Resulting cache behavior
0 (default)	Make the cache lifetime unlimited.
Positive integer	Set the maximum lifetime of the entire cache, in minutes, to the specified value. When the maximum lifetime for the cache is reached, the next attempt to read any entry from the cache causes the cache to be cleared

com.ibm.websphere.naming.jndicache.maxentrylife

Impose a limit to the age of individual cache entries with this property.

By default, cached objects remain in the cache for the life of the process or until cleared with the com.ibm.websphere.naming.jndicache.cacheobject property set to `cleared`. This property enables a JNDI client to set the maximum lifetime of individual cache entries. This property differs from the maxcachelife property in that individual entries are refreshed individually as their maximum lifetime reached. This might avoid any noticeable change in performance that might occur if the whole cache is cleared at once. The table below lists the various maxentrylife values and their effect on cache behavior:

Valid options	Resulting cache behavior
0 (default)	Lifetime of cache entries is unlimited.
Positive integer	Set the maximum lifetime of individual cache entries, in minutes, to the specified value. When the maximum lifetime for an entry is reached, the next attempt to read the entry from the cache causes the individual cache entry to refresh.

Example: Controlling JNDI cache behavior from a program

You can specify JNDI cache properties in a program to control the behavior of a JNDI cache.

Following are examples that illustrate how you can use JNDI cache properties to achieve the desired cache behavior. Cache properties take effect when an InitialContext object is constructed.

```
import java.util.Hashtable;
import javax.naming.InitialContext;
import javax.naming.Context;
import com.ibm.websphere.naming.PROPS;

/*****
 Caching discussed in this section pertains to the WebSphere Application
 Server initial context factory. Assume the property,
 java.naming.factory.initial, is set to
 "com.ibm.websphere.naming.WsnInitialContextFactory" as a
 java.lang.System property.
*****/

Hashtable env;
Context ctx;

// To clear a cache:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_CLEARED);
ctx = new InitialContext(env);

// To set a cache's maximum cache lifetime to 60 minutes:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_MAX_LIFE, "60");
ctx = new InitialContext(env);

// To turn caching off:

env = new Hashtable();
```

```
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE);
ctx = new InitialContext(env);

// To use caching and no caching:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_POPULATED);
ctx = new InitialContext(env);
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE);
Context noCacheCtx = new InitialContext(env);

Object o;

// Use caching to look up home, since the home should rarely change.
o = ctx.lookup("com/mycom/MyEJBHome");
// Narrow, etc. ...

// Do not use cache if data is volatile.
o = noCacheCtx.lookup("com/mycom/VolatileObject");
// ...
```

JNDI name syntax

JNDI name syntax is the default syntax and is suitable for typical JNDI clients.

This syntax includes the following special characters: forward slash (/) and backslash (\). Components in a name are delimited by a forward slash. The backslash is used as the escape character. A forward slash is interpreted literally if it is escaped, that is, preceded by a backslash. Similarly, a backslash is interpreted literally if it is escaped.

INS name syntax

INS syntax is designed for JNDI clients that need to interoperate with CORBA applications.

The INS syntax allows a JNDI client to make the proper mapping to and from a CORBA name. INS syntax is very similar to the JNDI syntax with the additional special character, dot (`.`). Dots are used to delimit the `id` and `kind` fields in a name component. A dot is interpreted literally when it is escaped. Only one unescaped dot is allowed in a name component. A name component with a non-empty `id` field and empty `kind` field is represented with only the `id` field value and must not end with an unescaped dot. An empty name component (empty `id` and empty `kind` field) is represented with a single unescaped dot. An empty string is not a valid name component representation.

JNDI to CORBA name mapping considerations

WebSphere Application Server name servers are an implementation of the CORBA CosNaming interface. WebSphere Application Server provides a JNDI implementation which you can use to access CosNaming name servers through the JNDI interface. Issues can exist when mapping JNDI name strings to and from CORBA names.

Each component in a CORBA name consists of an `id` and `kind` field, but a JNDI name component consists of no such fields. Each component in a JNDI name is atomic. Typical JNDI clients do not need to make a distinction between the `id` and `kind` fields of a name component, or know how JNDI name strings map to CORBA names. JNDI clients of this sort can use the JNDI syntax described below. When a name is parsed according to JNDI syntax, each name component is mapped to the `id` field of the corresponding CORBA name component. The `kind` field always has an empty value. This basic syntax is the least obtrusive to the JNDI client in that it has the fewest special characters. However, you cannot represent with this syntax a CORBA name with a non-empty `kind` field. This restriction can prevent EJB applications from interoperating with CORBA applications.

Some clients, however must interoperate with CORBA applications which use CORBA names with non-empty `kind` fields. These JNDI clients must make a distinction between `id` and `kind` so that JNDI

names are correctly mapped to CORBA names, particularly when the CORBA names contain components with non-empty `kind` fields. Such JNDI clients can use the INS name syntax. With its additional special character, you can use INS to represent any CORBA name. Use of this syntax is not recommended unless it is necessary, because this syntax is more restrictive from the JNDI client's perspective in that the JNDI client must be aware that name components with multiple unescaped dots are syntactically invalid. INS name syntax is part of the OMG CosNaming Interoperable Naming Specification.

Example: Setting the syntax used to parse name strings

JNDI clients that must interoperate with CORBA applications might need to use INS name syntax to represent names in string format.

The name syntax property can be passed to the InitialContext constructor through its parameter, in the System properties, or in a `jndi.properties` file. The initial context and any contexts looked up from that initial context parse name strings based on the specified syntax.

The following example shows how to set the name syntax to make the initial context parse name strings according to INS syntax.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS; // WebSphere naming constants
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, ...);
env.put(PROPS.NAME_SYNTAX, PROPS.NAME_SYNTAX_INS);
Context initialContext = new InitialContext(env);
// The following name maps to a CORBA name component as follows:
//    id = "a.name", kind = "in.INS.format"
// The unescaped dot is used as the delimiter.
// Escaped dots are interpreted literally.
java.lang.Object o = initialContext.lookup("a\.name.in\.INS\.format");
...
```

Chapter 18. Object Request Broker

Managing Object Request Brokers

Use this task to manage Object Request Brokers (ORB). An ORB manages the interaction between clients and servers using the Internet InterORB Protocol (IIOP).

Default property values are set when the product starts and the Java Object Request Broker (ORB) service is initialized. These properties control the run-time behavior of the ORB and can also affect the behavior of product components that are tightly integrated with the ORB, such as security. It might be necessary to modify some ORB settings under certain conditions.

Every request or response exchange consists of a client-side ORB and a server-side ORB. It is important to set the ORB properties for both sides as necessary.

After an ORB instance has been established in a process, changes to ORB properties do not affect the behavior of the running ORB instance. The process must be stopped and restarted for the modified properties to take effect.

A list of possible tasks for managing ORB follows. These steps can be performed in any order.

1. Adjust timeout settings to improve handling of network failures. See “Object Request Broker service settings” on page 1808 for more information. Before making these adjustments, read Object Request Broker tuning guidelines.
2. Adjust thread-pool settings used by the ORB for handling Internet InterORB Protocol (IIOP) connections. See Thread pool settings for more information.
3. If problems with the ORB arise, see “Object request broker troubleshooting tips” on page 1826.
   For help in troubleshooting, see “Object Request Broker communications trace” on page 1819.

Object Request Brokers

An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It enables clients to make requests and receive responses from servers in a network-distributed environment.

The ORB provides a framework for clients to locate objects in the network and to call operations on those objects as if the remote objects are located in the same running process as the client, providing location transparency. The client calls an operation on a local object, known as a *stub*. The stub forwards the request to the remote object, where the operation runs and the results are returned to the client.

The client-side ORB is responsible for creating an IIOP request that contains the operation and required parameters, and for sending the request on the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB demarshals the returned results and passes the result to the stub, which, in turn, returns to the client application, as if the operation had been run locally.

This product uses an ORB to manage communication between client applications and server applications as well as communication among product components. During product installation, default property values are set when the ORB is initialized. These properties control the run-time behavior of the ORB and can also affect the behavior of product components that are tightly integrated with the ORB, such as security. This product does not support the use of multiple ORB instances.

Logical pool distribution

The Logical pool distribution (LPD) thread pool mechanism implements a strategy for improving the performance of requests that have shorter run times. Do not configure LPD unless you have already configured it in a previous release of WebSphere Application Server. LPD is a deprecated function and will be removed in a future version of the product.

The need for LPD is indicated by a mixture of Enterprise JavaBeans (EJB) requests where the run times vary across the request types, and the ORB thread pool must be constrained for performance reasons. In this case, longer run time requests might tend to prolong the response times for shorter requests by denying them adequate access to threads in the thread pool. LPD provides a mechanism that allows shorter requests greater access to the threads.

LPD divides the Object Request Broker (ORB) thread pool into logical pools, as configured by the administrator using ORB custom properties starting that start with the following:

```
com.ibm.websphere.threadpool.strategy.*
```

The size of each pool is a percentage of the maximum number of ORB threads. The sum of the logical pool percentages must equal 100.

When LPD is active, incoming ORB requests are vectored, or pointed, to a pool based on historical run time history for the request type. The request type is determined by the method, which is qualified internally as unique across components. The LPD mechanism adjusts pool targets at runtime to optimize the distribution of requests across logical pools.

The LPD mechanism can be tuned after it is enabled. Response time, throughput measurements, and statistics produced by the LPD mechanism drive the tuning process.

Object Request Broker service settings

Use this page to configure the Java Object Request Broker (ORB) service.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Container services > ORB service** .

Several settings are available for controlling internal Object Request Broker (ORB) processing. You can use these settings to improve application performance in the case of applications that contain enterprise beans. You can make changes to these settings for the default server or any application server that is configured in the administrative domain.

Request timeout

Specifies the number of seconds to wait before timing out on a request message.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.RequestTimeout.

Data type	int
Units	Seconds
Default	180
Range	0 - largest integer recognized by Java

Request retries count

Specifies the number of times that the ORB attempts to send a request if a server fails. Retrying sometimes enables recovery from transient network failures. This field is ignored on the z/OS platform.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.requestRetriesCount.

Data type	int
Default	1
Range	1 to 10

Request retries delay

Specifies the number of milliseconds between request retries. This field is ignored on the z/OS platform.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.requestRetriesDelay.

Data type	int
Units	Milliseconds
Default	0
Range	0 to 60,000

Connection cache maximum

Specifies the maximum number of entries that can occupy the ORB connection cache before the ORB starts to remove inactive connections from the cache. This field is ignored on the z/OS platform.

It is possible that the number of active connections in the cache will temporarily exceed this threshold value. If necessary, the ORB will continue to add connections as long as resources are available.

For use in command-line scripting, the full name of this system property is com.ibm.CORBA.MaxOpenConnections.

Data type	Integer
Units	Connections
Default	240
Range	10 - largest integer recognized by Java

Connection cache minimum

Specifies the minimum number of entries in the ORB connection cache. This field is ignored on the z/OS platform.

The ORB will not remove inactive connections when the number of entries is below this value.

For use in command-line scripting, the full name of this system property is com.ibm.CORBA.MinOpenConnections.

Data type	Integer
Units	Connections
Default	100
Range	Any integer that is at least 5 less than the value specified for the Connection cache maximum property.

ORB tracing

Enables the tracing of ORB General Inter-ORB Protocol (GIOP) messages.

This setting affects two system properties: com.ibm.CORBA.Debug and com.ibm.CORBA.CommTrace. If you set these properties through command-line scripting, you must set both properties to `true` to enable the tracing of GIOP messages.

Data type	Boolean
Default	Not enabled (false)

Locate request timeout

Specifies the number of seconds to wait before timing out on a LocateRequest message. This field is ignored on the z/OS platform.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.LocateRequestTimeout.

Data type	int
Units	Seconds
Default	180
Range	0 to 300

Force tunneling

Controls how the client ORB attempts to use HTTP tunneling. This field is ignored on the z/OS platform.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.ForceTunnel.

Data type	String
Default	`NEVER`
Range	Valid values are `ALWAYS`, `NEVER`, or `WHENREQUIRED`.

Considering the following information when choosing the valid value:

ALWAYS
: Use HTTP tunneling immediately, without trying TCP connections first.

NEVER
: Disable HTTP tunneling. If a TCP connection fails, a CORBA system exception (COMM_FAILURE) occurs.

WHENREQUIRED
: Use HTTP tunneling if TCP connections fail.

Tunnel agent URL

Specifies the Web address of the servlet to use in support of HTTP tunneling. This field is ignored on the z/OS platform.

This Web address must be a proper format:

http://w3.mycorp.com:81/servlet/com.ibm.CORBA.services.IIOPTunnelServlet

For applets: http://applethost:port/servlet/com.ibm.CORBA.services.IIOPTunnelServlet.

This field is required if HTTP tunneling is set. If you use command-line scripting, the full name of this system property is com.ibm.CORBA.TunnelAgentURL.

Pass by reference

Specifies how the ORB passes parameters. If enabled, the ORB passes parameters by reference instead of by value, to avoid making an object copy. If you do not enable the pass by reference option, a copy of the parameter passes rather than the parameter object itself. This can be expensive because the ORB must first make a copy of each parameter object.

You can use this option only when the Enterprise JavaBeans (EJB) client and the EJB are on the same classloader. This requirement means that the EJB client and the EJB must be deployed in the same EAR file.

If the Enterprise JavaBeans (EJB) client and server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, enabling the pass by reference option can improve performance up to 50%. The pass by reference option helps performance only where non-primitive object types are passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

Important: Enable this property with caution because unexpected behavior can occur. If an object reference is modified by the callee, the caller's object is modified as well, since they are the same object.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.iiop.noLocalCopies.

Data type	Boolean
Default	Not enabled (false)

The use of this option for enterprise beans with remote interfaces violates Enterprise JavaBeans (EJB) Specification, Version 2.0 (see section 5.4). Object references passed to Enterprise JavaBeans (EJB) methods or to EJB home methods are not copied and can be subject to corruption.

Consider the following example:

```
Iterator iterator = collection.iterator();
MyPrimaryKey pk = new MyPrimaryKey();
while (iterator.hasNext()) {
   pk.id = (String) iterator.next();
   MyEJB myEJB = myEJBHome.findByPrimaryKey(pk);
}
```

In this example, a reference to the same MyPrimaryKey object passes into WebSphere Application Server with a different ID value each time. Running this code with pass by reference enabled causes a problem within the application server because multiple enterprise beans are referencing the same MyPrimaryKey object. To avoid this problem, set the com.ibm.websphere.ejbcontainer.allowPrimaryKeyMutation system property to `true` when the pass by reference option is enabled. Setting the pass by reference option to `true` causes the EJB container to make a local copy of the PrimaryKey object. As a result, however, a small portion of the performance advantage of setting the pass by reference option is lost.

As a general rule, any application code that passes an object reference as a parameter to an enterprise bean method or to an EJB home method must be scrutinized to determine if passing that object reference results in loss of data integrity or in other problems.

After examining your code, you can enable the pass by reference option by setting the com.ibm.CORBA.iiop.noLocalCopies system property to `true`. You can also enable the pass by reference option in the administrative console. Click **Servers > Application servers >** *server_name* **> Container services > ORB Service** and select **Pass by reference**.

Object Request Broker custom properties

There are several ways to configure an ORB. For example, you can use ORB custom property settings, or system property settings to configure an ORB, or you can provide objects during ORB initialization.

If you use ORB custom properties to configure an ORB, you must understand that there are two types of default values for some of these properties: JDK default values and WebSphere Application Server default

values. The JDK default is the value that the ORB uses for a property if the property is not specified in any way. The WebSphere Application Server default is the value that WebSphere Application Server sets for a property in one of the following files:
- The orb.properties file when an application server is installed.
- The server.xml when an application server is configured.

Because WebSphere Application Server explicitly sets its default value, if both a WebSphere Application Server and a JDK default value is defined for a property, the WebSphere Application Server default takes precedence over the JDK default.

For more information about the different ways to specify ORB properties and the precedence order, see the JDK Diagnostic Guide for the version of the JDK that you are using. These guides are available at http://www-128.ibm.com/developerworks/java/jdk/diagnosis/.

The WebSphere Application Server orb.properties file, that is located in the WebSphere Application Server *was_home*/java/jre/lib directory, contains WebSphere Application server ORB custom properties that are initially set to WebSphere Application Server default values during the WebSphere Application Server installation process.

You can specify new values for the ORB custom properties in the administrative console. Any value you specify takes precedence over any JDK or WebSphere Application Server default values for these properties. The ORB custom properties settings that you specify in the administrative console are stored in the WebSphere Application Server server.xml system file and are passed to an ORB in a properties object whenever an ORB is initialized.

To use the administrative console to set ORB custom properties, in the administrative console, click **Servers >Application servers >** *server_name* **>Container services > ORB service >Custom properties**. You can then change the setting of one of the listed custom properties or click **New** to add a new property to the list. Then click **Apply** to save your change. When you finish making changes, click **OK** and then click **Save** to save your changes.

To use the java command on a command line, use the -D option. For example:

```
java -Dcom.ibm.CORBA.propname1=value1 -Dcom.ibm.CORBA.propname2=value2 ... application name
```

To use the launchclient command on a command line, prefix the property with -CC. For example:

```
launchclient yourapp.ear -CCDcom.ibm.CORBA.propname1=value1 -CCDcom.ibm.CORBA.propname2=value2
   ... optional application arguments
```

The Custom properties page might already include Secure Sockets Layer (SSL) properties that were added during WebSphere Application Server installation. A list of additional properties associated with the Java ORB service follows. Unless otherwise indicated, the default values provided in the descriptions of these properties are the JDK default values.

com.ibm.CORBA.BootstrapHost

Specifies the domain name service (DNS) host name or IP address of the machine on which initial server contact for this client resides. This setting is deprecated and is scheduled for removal in a future release.

For a command-line or programmatic alternative, see “Client-side programming tips for the Java Object Request Broker service” on page 1822.

com.ibm.CORBA.BootstrapPort

Specifies the port to which the ORB connects for bootstrapping, the port of the machine on which the initial server contact for this client listens. This setting is deprecated and is scheduled for removal in a future release.

For a command-line or programmatic alternative, see “Client-side programming tips for the Java Object Request Broker service” on page 1822.

Default	2809

com.ibm.CORBA.ConnectTimeout

The com.ibm.CORBA.ConnectTimeout property specifies the maximum time in seconds that the client ORB waits before timing out when attempting to establish an IIOP connection with a remote server ORB. Generally, client applications use this property. The property is not used by the application server by default. However, if necessary, you can specify the property for each individual application server through the administrative console.

Client applications can specify the com.ibm.CORBA.ConnectTimeout property in one of two ways:

- By including it in the orb.properties file.
- By using the -CCD option to set the property with the launchclient script. This example specifies a maximum timeout value of ten seconds:

```
launchclient clientapp.ear -CCDcom.ibm.com.CORBA.ConnectTimeout=10...
```

Begin by setting your timeout value to 20-30 seconds, but consider factors such as network congestion and application server load and capacity. Lower values can provide better failover performance, but can result in exceptions if the remote server does not have enough time to complete the connection.

Valid Range	0-300 (seconds)
Default	0 (the client ORB waits indefinitely)

com.ibm.CORBA.ConnectionInterceptorName

Specifies the connection interceptor class that is used to determine the type of outbound IIOP connection to use for a request, and if secure, the quality of protection characteristics associated with the request.

WebSphere Application Server default	com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor
JDK default	None.

com.ibm.CORBA.enableLocateRequest

Specifies whether the ORB uses the locate request mechanism to find objects in a WebSphere Application Server cell. Use this property for performance tuning.

When this property is set to true, the ORB first sends a short message to the server to find the object that it needs to access. This first contact is called the *locate request*. If most of your initial method invocations are very small, setting this property to false might improve performance because this setting change can reduce the GIOP traffic by as much as half. If most of your initial method invocations are large, you should set this property to true so that the small locate request message is sent instead of the large message. The large message is then sent to the proper target after the desired object is found.

WebSphere Application Server default	true
JDK default	false

com.ibm.CORBA.FragmentSize

Specifies the size of General Inter-ORB Protocol (GIOP) fragments used by the ORB. If the total size of a request exceeds the set value, the ORB breaks up and sends multiple fragments until the entire request is sent. Set this property on the client side with a -D system property if you use a stand-alone Java application.

Adjust the `com.ibm.CORBA.FragmentSize` property if the amount of data that is sent over Internet Inter-ORB Protocol (IIOP) in most General Inter-ORB Protocol (GIOP) requests exceeds one kilobyte or if thread dumps show that most client-side threads seem to be waiting while sending or receiving data. Adjust this property so that most messages have few or no fragments.

If you want to instruct the ORB not to break up any of the requests or replies it sends, set this property to `0` (zero). However, setting the value to zero does not prevent the ORB from receiving GIOP fragments in requests or replies sent by another existing ORB.

Units	Bytes.
Default	1024
Range	From 64 to the largest value of a Java integer type that is divisible by 8

com.ibm.CORBA.ListenerPort

Specifies the port on which this server listens for incoming requests. The setting of this property is valid for client-side ORBs only.

Default	Next available system-assigned port number
Range	0 to 2147483647

com.ibm.CORBA.LocalHost

Specifies the host name or IP address of the system on which the server ORB is running. If you do not specify a value for this property, during ORB initialization WebSphere Application Server sets this property to the host name or IP address specified for the BOOTSTRAP_ADDRESS end point in serverindex.xml. For client applications, if no value is specified for this property, the ORB obtains a value at run time by calling InetAddress.getLocalHost().getHostAddress() method.

com.ibm.CORBA.numJNIReaders

Specifies the number of JNI reader threads to be allocated in the ORB's JNI reader thread pool. Each thread can handle up to 1024 connections.

Attention: Before setting this property, make sure that a JSSE provider has been selected as the provider for the SSL repertoire associated with the port on which the ORB service listens for incoming requests. IBMJSSE2 is the default provider setting for SSL repertoires and does not provide the file descriptor that JNIReader Threads require.

Valid Range	1-2147483647
Default	4

com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.JNIReaderPoolImpl

Specifies that JNI reader threads will be used. The property name specifies the class name of the ORB component that manages the pool of JNI reader threads and interacts with the native OS library used to process multiple connections simultaneously.

Attention:

- Make sure the library is in the WebSphere Application Server bin directory.

  For an Intel platform, the library is Selector.dll and for a UNIX platform, it is libSelector.a or libSelector.so.

  For the UNIX platform, if the prefix ″lib″ is missing, the file should be renamed.
- When specifying this property using the administrative console, enter com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.JNIReaderPoolImpl for the property name and an empty string (″″) for the value.

  When specifying this property on the java command, do not include a value:

  ```
  -Dcom.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.JNIReaderPoolImpl
  ```

Valid Range	not applicable
Default	none

com.ibm.CORBA.RasManager

Specifies an alternative to the default RAS manager of the ORB. This property must be set to com.ibm.websphere.ras.WsOrbRasManager before the ORB can be integrated with the rest of WebSphere Application Server RAS processing.

WebSphere Application Server default	com.ibm.websphere.ras.WsOrbRasManager
JDK default	None.

com.ibm.CORBA.ServerSocketQueueDepth

Specifies the maximum number of connection requests that can remain unhandled by the WebSphere Application Server ORB before the Application Server starts to reject new incoming connection requests. This property corresponds to the `backlog` argument to a ServerSocket constructor and is handled directly by TCP/IP.

If you see a ″connection refused″ message in a trace log, usually either the port on the target machine isn't open, or the server is overloaded with queued-up connection requests. Increasing the value specified for this property can help alleviate this problem if there does not appear to be any other problem in the system.

Default	50
Range	From 50 to the largest value of the Java int type

com.ibm.CORBA.ShortExceptionDetails

Specifies that the exception detail message that is returned whenever the server ORB encounters a CORBA system exception contains a short description of the exception as returned by the toString method of java.lang.Throwable class. Otherwise, the message contains the complete stack trace as returned by the printStackTrace method of java.lang.Throwable class.

com.ibm.CORBA.WSSSLClientSocketFactoryName

Specifies the class that the ORB uses to create SSL sockets for secure outbound IIOP connections.

WebSphere Application Server dfault	com.ibm.ws.security.orbssl.WSSSLClientSocketFactoryImpl
JDK default	None.

com.ibm.CORBA.WSSSLServerSocketFactoryName

Specifies the class that the ORB uses to create SSL sockets for inbound IIOP connections.

WebSphere Application Server default	com.ibm.ws.security.orbssl.WSSSLServerSocketFactoryImpl
JDK default	None.

com.ibm.websphere.ObjectIDVersionCompatibility

This property applies when you have a mixed release cluster for which you are performing an incremental cell upgrade, and at least one of the releases is earlier than V5.1.1.

In an environment that includes mixed release cells, the migration program automatically sets this property to 1.

After you upgrade all of the cluster members to the same release, you can removing this property from the list of ORB custom properties or you can change the value that is specified for the property to 2. Either action improves performance.

When this property is set to 1, the ORB runs using version 1 object identities, which are required for mixed cells that contain application servers with releases prior to V5.1.1. If you do not specify a value for this property or if you set this property to 2, the ORB runs using version 2 object identities, which cannot be used with pre-V5.1.1 application servers.

See the *Migrating, coexisting, and interoperating* PDF for instructions on how to perform an incremental cell upgrade.

com.ibm.websphere.threadpool.strategy.implementation

Specifies the logical pool distribution (LPD) thread pool strategy the next time you start the application server, and is enabled if set to com.ibm.ws.threadpool.strategy.LogicalPoolDistribution.

Attention: This is a deprecated function and will be removed in a future version of the product. Do not configure logical pool distribution unless you have already configured it for a previous release of WebSphere Application Server.

Some requests have shorter start times than others. LPD is a mechanism for providing these shorter requests greater access to start threads. For more information, see the information center.

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.calcinterval

Specifies how often the logical pool distribution (LPD) mechanism readjusts the pool start target times. This property cannot be turned off after this support is installed.

Attention: Do not configure logical pool distribution unless you have already configured it with a previous release of WebSphere Application Server. This function is deprecated and will be removed in a future version of the product.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Units	Milliseconds
Default	30
Range	20,000 milliseconds minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.lruinterval

Specifies how long the logical pool distribution internal data is kept for inactive requests. The mechanism tracks several statistics for each request type that is received. Consider removing requests that have been inactive for awhile.

Attention: Do not configure logical pool distribution unless you have already configured it with a previous release of WebSphere Application Server. This function is deprecated and will be removed in a future version of the product.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Units	Milliseconds
Default	300,000 (5 minutes)
Range	60,000 (1 minute) minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.outqueues

Specifies how many pools are created and how many threads are allocated to each pool in the logical pool distribution mechanism.

Attention: Do not configure logical pool distribution unless you have already configured it with a previous release of WebSphere Application Server. This function is deprecated and will be removed in a future version of the product.

The ORB parameter for max threads controls the total number of threads. The outqueues parameter is specified as a comma separated list of percentages that add up to 100. For example, the list 25,25,25,25 sets up 4 pools, each allocated 25% of the available ORB thread pool. The pools are indexed left to right from 0 to n-1. Each outqueue is dynamically assigned a target start time by the calculation mechanism. Target start times are assigned to outqueues in increasing order so pool 0 gets the requests with the least start time and pool n-1 gets requests with the highest start times.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integers in comma separated list
Default	25,25,25,25
Range	Percentages in list must total 100 percent

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.statsinterval

Specifies that statistics are dumped to stdout after this interval expires, but only if requests are processed. This process keeps the mechanism from filling the log files with redundant information. These statistics are beneficial for tuning the logical pool distribution mechanism.

Attention: Do not configure logical pool distribution unless you have already configured it with a previous release of WebSphere Application Server. This function is deprecated and will be removed in a future version of the product.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Units	Milliseconds
Default	0 (off)
Range	30,000 (30 seconds) minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.workqueue

Specifies the size of a new queue where incoming requests wait for dispatch. Pertains to the logical pool distribution mechanism.

Attention: Do not configure logical pool distribution unless you have already configured it with a previous release of WebSphere Application Server. This function is deprecated and will be removed in a future version of the product.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Default	96
Range	10 minimum

com.ibm.ws.orb.services.redirector.MaxOpenSocketsPerEndpoint

Specifies the maximum number of connections that the IIOP Tunnel Servlet should maintain in its connection cache for each target host or port. If the number of concurrent client requests to a single host or port exceeds the setting for this property, the IIOP Tunnel Servlet opens a temporary connection to the target server for each extra client request, and then closes the connection after it receives the reply. Connections that are opened but not used within five minutes are removed from the cache for the IIOP Tunnel Servlet.

WebSphere Application Server default	3
JDK default	Not applicable
Range	0 - largest integer recognized by Java

com.ibm.ws.orb.services.redirector.RequestTimeout

Specifies the number of seconds that the IIOP Tunnel Servlet waits for a reply from the target server on behalf of a client before timing out. If a value is not specified for this property, or is improperly specified, the com.ibm.CORBA.RequestTimeout property setting for the application server on which the IIOP Tunnel Servlet is installed is used as the setting for the com.ibm.ws.orb.services.redirector.RequestTimeout property.

The value you specify for this property should be at least as high as the highest client setting for the com.ibm.CORBA.RequestTimeout property, otherwise the IIOP Tunnel Servlet might timeout more quickly than the client would normally timeout while waiting for a reply. If this property is set to zero, the IIOP Tunnel Servlet does not time out.

WebSphere Application Server default	com.ibm.CORBA.RequestTimeout property setting for the application server on which the IIOP Tunnel Servlet is installed.
JDK default	Not applicable
Range	0 - largest integer recognized by Java

com.ibm.ws.orb.transport.useMultiHome

Specifies whether the WebSphere Application Server ORB binds to all network interfaces in the system. If true is specified, the ORB binds to all network interfaces that are available to it. If false is specified, the ORB only binds to the network interface specified for the com.ibm.CORBA.LocalHost system property.

WebSphere Application Server default	true
JDK default	true

javax.rmi.CORBA.UtilClass

Specifies the name of the Java class that WebSphere Application Server uses to implement the javax.rmi.CORBA.UtilDelegate interface.

This property supports delegation for method implementations in the javax.rmi.CORBA.Util class. The javax.rmi.CORBA.Util class provides utility methods that can be used by stubs and ties to perform common operations. The delegate is a singleton instance of a class that implements this interface and provides a replacement implementation for all of the methods of javax.rmi.CORBA.Util. To enable a delegate, provide the class name of the delegate as the value of the javax.rmi.CORBA.UtilClass system property. The default value provides support for the com.ibm.CORBA.iiop.noLocalCopies property.

WebSphere Application Server default	com.ibm.ws.orb.WSUtilDelegateImpl
JDK default	None.

Object Request Broker communications trace

The Object Request Broker (ORB) communications trace, typically referred to as *CommTrace*, contains the sequence of General InterORB Protocol (GIOP) messages sent and received by the ORB when the application is running.

It might be necessary to understand the low-level sequence of client-to-server or server-to-server interactions during problem determination. This topic uses trace entries from log examples to explain the contents of the log and help you understand the interaction sequence. It focuses only in the GIOP messages and does not discuss in detail additional trace information that displays when intervening with the GIOP-message boundaries.

Location

When ORB tracing is enabled, this information is placed in the *app_server_root*/profiles/*profile_name*/logs/*server_name*/trace.log directory.

About the ORB trace file

The following are properties of the file that is created when ORB tracing is enabled.
- Read-only
- Updated by the administrative function
- Use this file to localize and resolve ORB-related problems.

How to interpret the output

The following sections refer to sample log output found later in this topic.

Identifying information

The start of a GIOP message is identified by a line that contains either OUT GOING: or IN COMING: depending on whether the message is sent or received by the process.

Following the identifying line entry is a series of items, formatted for convenience, with information extracted from the raw message that identifies the endpoints in this particular message interaction. See lines 3-13 in both examples. The formatted items include the following:
- GIOP message type (line 3)
- Date and time that the message was recorded (line 4)
- Information that is useful to identify the thread that is running when the message records, and other thread-specific information (line 5)
- Local and remote TCP/IP ports used for the interaction (lines 6 through 9)
- GIOP version, byte order, an indication of whether the message is a fragment, and message size (lines 10 through 13)

Request ID, response expected and reply status

Following the introductory message information, the request ID is an integer generated by the ORB. It is used to identify and associate each request with its corresponding reply. This association is necessary because the ORB can receive requests from multiple clients and must be able to associate each reply with the corresponding originating request.
- Lines 15-17 in the request example show the request ID, followed by an indication to the receiving endpoint that a response is expected (CORBA supports sending one-way requests for which a response is not expected.)
- Line 15 in the *Sample Log Entry - GIOP Reply* shows the request ID; line 33 shows the reply status received after completing the previously sent request.

Object Key

Lines 18-20 in the request example show the object key, the internal representation used by the ORB to identify and locate the target object intended to receive the request message. Object keys are not standardized.

Operation

Line 21 in the request example shows the name of the operation that the target object starts in the receiving endpoint. In this example, the specific operation requested is named _get_value.

Service context information

The service contexts in the message are also formatted for convenience. Each GIOP message might contain a sequence of service contexts sent and received by each endpoint. Service contexts, identified uniquely with an ID, contain data used in the specific interaction, such as security, character code set conversion, and ORB version information. The content of some of the service contexts is standardized and specified by OMG, while other service contexts are proprietary and specified by each vendor. IBM-specific service contexts are identified with IDs that begin with 0x4942.

Lines 22-41 in the request example illustrate typical service context entries. Three service contexts are in the request message, as shown in line 22. The ID, length of data, and raw data for each service context is printed next. Lines 23-25 show an IBM-proprietary context, as indicated by the 0x49424D12 ID. Lines 26-41 show two standard service contexts, identified by 0x6 ID (line 26) and the 0x1 ID (line 39).

Lines 16-32 in the *Sample Log Entry - GIOP Reply* illustrate two service contexts, one IBM-proprietary (line 17) and one standardized (line 20).

For the definition of the standardized service contexts, see the CORBA specification. Service context 0x1 (CORBA::IOP::CodeSets) is used to publish the character code sets supported by the ORB in order to negotiate and determine the code set used to transmit character data. Service context 0x6 (CORBA::IOP::SendingContextRunTime) is used by Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP) to provide the receiving endpoint with the IOR for the SendingContextRuntime object. IBM service context 0x49424D12 is used to publish ORB PartnerVersion information to support release-to-release interoperability between sending and receiving ORBs.

Data offset

Line 42 in the request example shows the offset, relative to the beginning of the GIOP message, where the remainder body of the request or reply message is located. This portion of the message is specific to each operation and varies from operation to operation. Therefore, it is not formatted, as the specific contents are not known by the ORB. The offset is printed as an aid to quickly locating the operation-specific data in the raw GIOP message dump, which follows the data offset.

Raw GIOP message dump

Starting at line 45 in the request example and line 36 in the *Sample Log Entry - GIOP Reply*, a raw dump of the entire GIOP message is printed in hexadecimal format. Request messages contain the parameters required by the given operation and reply messages contain the return values and content of output parameters as required by the given operation. For brevity, not all of the raw data is in the figures.

Sample Log Entry - GIOP Request

```
1.  OUT GOING:

3.  Request Message
4.  Date:          April 17, 2002 10:00:43 PM CDT
5.  Thread Info:   P=842115:O=1:CT
6.  Local Port:    1243 (0x4DB)
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10.  GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
```

```
13. Message size:  268 (0x10C)
--
15. Request ID:        5
16. Response Flag:     WITH_TARGET
17. Target Address:     0
18. Object Key:        length = 24 (0x18)
                       4B4D4249 00000010 BA4D6D34 000E0008
                       00000000 00000000
21. Operation:         _get_value
22. Service Context:   length = 3 (0x3)
23. Context ID:  1229081874 (0x49424D12)
24. Context data:  length = 8 (0x8)
                       00000000 13100003
26. Context ID:  6 (0x6)
27. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
                       6F72672F 53656E64 696E6743 6F6E7465
                       78742F43 6F646542 6173653A 312E3000
                       00000001 00000000 00000068 00010200
                       0000000E 3139322E 3136382E 312E3130
                       310004DC 00000018 4B4D4249 00000010
                       BA4D6D69 000E0008 00000000 00000000
                       00000002 00000001 00000018 00000000
                       00010001 00000001 00010020 00010100
                       00000000 49424D0A 00000008 00000000
                       13100003
39. Context ID:  1 (0x1)
40. Context data:  length = 12 (0xC)
                       00000000 00010001 00010100
42. Data Offset:       118


45. 0000: 47494F50 01020000 0000010C 00000005   GIOP............
46. 0010: 03000000 00000000 00000018 4B4D4249   ............KMBI
47. 0020: [remainder of message body deleted for brevity]
```

Sample Log Entry - GIOP Reply

```
1.  IN COMING:

3.  Reply Message
4.  Date:          April 17, 2002 10:00:47 PM CDT
5.  Thread Info:   RT=0:P=842115:O=1:com.ibm.rmi.transport.TCPTransportConnection
5a (line 5 broken for publication).   remoteHost=192.168.1.101 remotePort=1242 localPort=1243
6.  Local Port:    1243 (0x4DB)
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10. GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
13. Message size:  208 (0xD0)
--
15. Request ID:        5
16. Service Context:   length = 2 (0x2)
17. Context ID:  1229081874 (0x49424D12)
18. Context data:  length = 8 (0x8)
                       00000000 13100003
20. Context ID:  6 (0x6)
21. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
                       6F72672F 53656E64 696E6743 6F6E7465
                       78742F43 6F646542 6173653A 312E3000
                       00000001 00000000 00000068 00010200
                       0000000E 3139322E 3136382E 312E3130
                       310004DA 00000018 4B4D4249 00000010
                       BA4D6D34 000E0008 00000001 00000000
```

```
                      00000002 00000001 00000018 00000000
                      00010001 00000001 00010020 00010100
                      00000000 49424D0A 00000008 00000000
                      13100003
33. Reply Status:     NO_EXCEPTION


36. 0000: 47494F50 01020001 000000D0 00000005   GIOP............
37. 0010: 00000000 00000002 49424D12 00000008   ........IBM.....
38. 0020: [remainder of message body deleted for brevity]
```

Client-side programming tips for the Java Object Request Broker service

This topic includes programming tips for applications that communicate with the client-side Object Request Broker (ORB) that is part of the Java ORB service.

Resolution of initial references to services

Client applications can use the ORBInitRef and ORBDefaultInitRef properties to configure the network location that the Java ORB service uses to find a service such as naming. When set, these properties are included in the parameters that are used to initialize the ORB, as illustrated in the following example:

```
org.omg.CORBA.ORB.init(java.lang.String[] args,
                       java.util.Properties props)
```

You can set these properties in client code or by command-line argument. It is possible to specify more than one service location by using multiple ORBInitRef property settings (one for each service), but only a single ORBDefaultInitRef value can be specified.

For setting in client code, these properties are com.ibm.CORBA.ORBInitRef.*service_name* and com.ibm.CORBA.ORBDefaultInitRef, respectively. For example, to specify that the naming service (NameService) is located in sample.server.com at port 2809, set the com.ibm.CORBA.ORBInitRef.NameService property to corbaloc::sample.server.com:2809/NameService.

For setting by command-line argument, these properties are -ORBInitRef and -ORBDefaultInitRef, respectively. To locate the same naming service specified previously, use the following Java command:

After these properties are set for services that the ORB supports, Java 2 Platform, Enterprise Edition (J2EE) applications can call the resolve_initial_references function on the ORB, as defined in the CORBA/IIOP specification, to obtain the initial reference to a given service.

Preferred API for obtaining an ORB instance

For J2EE applications, you can use either of the following approaches. However, it is strongly recommended that you use the Java Naming and Directory Interface (JNDI) approach to ensure that the same ORB instance is used throughout the client application; you avoid the unintended inconsistencies that might occur when different ORB instances are used.

JNDI approach: For J2EE applications (including enterprise beans, J2EE clients and servlets), you can obtain an ORB instance by creating a JNDI InitialContext object and looking up the ORB under the java:comp/ORB name , as illustrated in the following example:

```
javax.naming.Context ctx = new javax.naming.InitialContext();
org.omg.CORBA.ORB orb =
   (org.omg.CORBA.ORB)javax.rmi.PortableRemoteObject.narrow(ctx.lookup("java:comp/ORB"),
                                                            org.omg.CORBA.ORB.class);
```

The ORB instance obtained using JNDI is a singleton object, shared by all the J2EE components that are running in the same Java virtual machine process.

CORBA approach: Because thin-client applications do not run in a J2EE container, they cannot use JNDI interfaces to look up the ORB. In this case, you can obtain an ORB instance by using CORBA programming interfaces, as follows:

```
java.util.Properties props = new java.util.Properties();
java.lang.String[] args = new java.lang.String[0];
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
```

In contrast to the JNDI approach, the CORBA specification requires that a new ORB instance be created each time the ORB.init method is called. If necessary to change the ORB default settings, you can add ORB property settings to the Properties object that is passed in the ORB.init method call.

The use of the com.ibm.ejs.oa.EJSORB.getORBinstance method, supported in previous releases of this product is deprecated.

API restrictions with sharing an ORB instance among J2EE application components

For performance reasons, it often makes sense to share a single ORB instance among components in a J2EE application. As required by the J2EE Specification, Version 1.3, all Web and EJB containers provide an ORB instance in the JNDI namespace as java:comp/ORB. Each container can share this instance among application components but is not required to. For proper isolation between application components, application code must comply with the following restrictions:

- Do not call the ORB shutdown or destroy methods
- Do not call org.omg.CORBA_2_3.ORB methods register_value_factory, or unregister_value_factory

In addition, do not share an ORB instance among application components in different J2EE applications.

Required use of rmic and idlj that ship with the IBM Developer Kit

The Java Runtime Environment (JRE) used by this product includes the **rmic** and **idlj** tools. You use the tools to generate Java language bindings for the CORBA/IIOP protocol.

During product installation, the tools are installed in the *app_server_root*/java/ibm_bin directory. Versions of these tools included with Java development kits in the $JAVA_HOME/bin directory other than the IBM Developer Kit installed with this product are incompatible with this product.

When you install this product, the *app_server_root*/java/ibm_bin directory is included in the $PATH search order to enable use of the rmic and idlj scripts provided by IBM. Because the scripts are in the *app_server_root*/java/ibm_bin directory instead of the JRE standard *app_server_root*/java/bin directory, it is unlikely that you can overwrite them when applying maintenance to a JRE not provided by IBM.

In addition to the rmic and idlj tools, the JRE also includes Interface Definition Language (IDL) files. The files are based on those defined by the Object Management Group (OMG) and can be used by applications that need an IDL definition of selected ORB interfaces. The files are placed in the *app_server_root*/java/ibm_lib directory.

Before using either the rmic or idlj tool, ensure that the *app_server_root*/java/ibm_bin directory is included in the proper PATH variable search order in the environment. If your application uses IDL files in the *app_server_root*/java/ibm_lib directory, also ensure that the directory is included in the PATH variable.

Character code set conversion support for the Java Object Request Broker service

The CORBA/IIOP specification defines a framework for negotiation and conversion of character code sets used by the Java Object Request Broker (ORB) service.

This product supports the framework and provides the following system properties for modifying the default settings:

com.ibm.CORBA.ORBCharEncoding

Specifies the name of the native code set that the ORB uses for character data (referred to as *NCS-C* in the CORBA/IIOP specification). By default, the ORB uses UTF8. Valid code set values for this property are shown in the table that follows this list; values that are valid only for ORBWCharDefault are indicated.

com.ibm.CORBA.ORBWCharDefault

Specifies the default code set that the ORB uses for transmission of wide character data when no code set for wide character data is found in the tagged component in the Interoperable Object Reference (IOR) or in the GIOP service context. If no code set for wide character data is found and this property is not set, the ORB raises an exception, as specified in the CORBA specification. No default value is set for this property. The only valid code set values for this property are UCS2 or UTF16.

The CORBA code set negotiation and conversion framework specifies the use of code set registry IDs as defined in the Open Software Foundation (OSF) code set registry. The ORB translates the Java file.encoding names shown in the following table to the corresponding OSF registry IDs. These IDs are then used by the ORB in the IOR Code set tagged component and GIOP code set service context as specified in the CORBA and IIOP specification.

Java name	OSF registry ID	Comments
ASCII	0x00010020	
ISO8859_1	0x00010001	
ISO8859_2	0x00010002	
ISO8859_3	0x00010003	
ISO8859_4	0x00010004	
ISO8859_5	0x00010005	
ISO8859_6	0x00010006	
ISO8859_7	0x00010007	
ISO8859_8	0x00010008	
ISO8859_9	0x00010009	
ISO8859_15_FDIS	0x0001000F	
Cp1250	0x100204E2	
Cp1251	0x100204E3	
Cp1252	0x100204E4	
Cp1253	0x100204E5	
Cp1254	0x100204E6	
Cp1255	0x100204E7	
Cp1256	0x100204E8	
Cp1257	0x100204E9	
Cp943C	0x100203AF	
Cp943	0x100203AF	
Cp949C	0x100203B5	
Cp949	0x100203B5	
Cp1363C	0x10020553	
Cp1363	0x10020553	

Java name	OSF registry ID	Comments
Cp950	0x100203B6	
Cp1381	0x10020565	
Cp1386	0x1002056A	
EUC_JP	0x00030010	
EUC_KR	0x0004000A	
EUC_TW	0x00050010	
Cp964	0x100203C4	
Cp970	0x100203CA	
Cp1383	0x10020567	
Cp33722C	0x100283BA	
Cp33722	0x100283BA	
Cp930	0x100203A2	
Cp1047	0x10020417	
UCS2	0x00010100	Valid only for the ORBWCharDefault
UTF8	0x05010001	
UTF16	0x00010109	Valid only for the ORBWCharDefault

Object Request Brokers: Resources for learning

Use the following links to find relevant supplemental information about Object Request Brokers (ORBs). The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- “Planning, business scenarios, and IT architecture”
- “Administration”
- “Programming specifications” on page 1826

Planning, business scenarios, and IT architecture

- CORBA FAQ

  Getting started with Object Request Brokers and CORBA.
- WebSphere Application Server CORBA Interoperability

  This document describes WebSphere CORBA interoperability for WebSphere Application Server products.
- CORBA Interoperability Samples

  These samples demonstrate the general principles by which WebSphere Application Server applications can interoperate with CORBA applications.

Administration

- IANA Character Set Registry

  This document contains a list of all valid character encoding schemes.
- developerWorks WebSphere

Programming specifications

- Catalog Of OMG CORBA/IIOP Specifications

  This document provides a catalog of OMG CORBA/IIOP specifications.

Object request broker troubleshooting tips

Use these tips to diagnose problems related to the WebSphere Application Server Object Request Broker (ORB).

Enabling tracing for the Object Request Broker component

The object request broker (ORB) service is one of the WebSphere Application Server run time services. Tracing messages sent and received by the ORB is a useful starting point for troubleshooting the ORB service. You can selectively enable or disable tracing of ORB messages for each server in a WebSphere Application Server installation, and for each application client.

This tracing is referred to by WebSphere Application Server support as a *comm trace*, and is different from the general purpose trace facility. The trace facility, which shows the detailed run-time behavior of product components, may be used alongside comm trace for other product components, or for the ORB component. The trace string associated with the ORB service is `ORBRas=all=enabled`.

You can enable and disable comm tracing using the administrative console or by manually editing the server.xml file for the server be trace. You must stop and restart the server for the configuration change to take effect.

For example, using the administrative console:

- Click **Servers > Application servers >** *server_name* **> Container services > ORB service**, and select the ORB tracing. Click **OK**, and then click **Save** to save your settings. Restart the server for the new settings to take effect. Or,
- Locate the server.xml file for the selected server, for example: *profile_root*/config/cells/*node_name*/nodes/*node_name*/servers/*server_name*/server.xml.
- Locate the services entry for the ORB service, `xmi:type=orb:ObjectRequestBroker`, and set `commTraceEnabled=true`.

ORB tracing for client applications requires that both the ORBRas component trace and the ORB comm trace are enabled. If only the ORB comm trace is enabled, no trace output is generated for client-side ORB traces. You can use one of the following options to enable both traces in the command line used to launch the client application:

- If you are using the WebSphere Application Server launcher, launchClient, use the **-CCD** option.
- If you are using the **java** command specify these parameters:

```
-trace=ORBRas=all=enabled
-tracefile=filename
-Dcom.ibm.CORBA.Debug=true
-Dcom.ibm.CORBA.CommTrace=true
```

ORB tracing output for thin clients can be directed by setting the `com.ibm.CORBA.Debug.Output = debugOutputFilename` parameter in the command line.

Important: When using launchClient on operating systems like AIX or Linux, because ORB tracing is integrated with the WebSphere Application Server general component trace, both the

component and comm ORB traces are written to the same file as the rest of the WebSphere Application Server traces. The name of this file is specified on the `-CCtracefile=`*filename* parameter.

When using launchClient on a Windows operating systems, you get a separate ORB trace file, called orbtrc.*timestamp*.txt. This file is located in the current directory.

Log files and messages associated with Object Request Broker

Messages and trace information for the ORB are saved in the following logs:
- The *profile_root*/logs/*server_name*/trace.log file for output from communications tracing and tracing the behavior of the ORBRas component
- The JVM logs for each application server, for WebSphere Application Server error and warning messages

The following message in the SystemOut.log file indicates the successful start of the application server and its ORB service:

WSVR0001I: Server server1 open for e-business

When communications tracing is enabled, a message similar to the following example in the *profile_root*/logs/*server_name*/trace.log file, indicates that the ORB service has started successfully. The message also shows the start of a listener thread, which is waiting for requests on the specified local port.

com.ibm.ws.orbimpl.transport.WSTransport startListening(ServerConnectionData connectionData) P=693799:O=0:CT a new ListenerThread has been started for ServerSocket[addr=0.0.0.0/ 0.0.0.0,port=0,localport=1360]

When the `com.ibm.ejs.oa.*=all=enabled` parameter is specified, tracing of the Object Adapter is enabled. The following message in the trace.log indicates that the ORB service started successfully:

EJSORBImpl < initializeORB

The ORB service is one of the first services started during the WebSphere Application Server initialization process. If it is not properly configured, other components such as naming, and security are not likely to start successfully. This is obvious in the JVM logs or trace.log of the affected application server.

Adjusting object request broker timeout values

If Web clients that access Java applications running in the product environment are consistently experiencing problems with their requests, and the problem cannot be traced to other sources and addressed through other solutions, consider setting an ORB timeout value and adjusting it for your environment. A list of timeout scenarios follows:
- Web browsers vary in their language for indicating that they have timed out. Usually, the problem is announced as a connection failure or a no-path-to-server message.
- Set an ORB timeout value to less than the time after which a Web client eventually times out. Because it can be difficult to tell how long Web clients wait before timing out, setting an ORB timeout value requires experimentation. The ideal testing environment features some simulated network failures for testing the proposed setting value.
- Empirical results from limited testing indicate that 30 seconds is a reasonable starting value. Ensure that this setting is not too low. To fine tune the setting, find a value that is not too low. Gradually decrease the setting until reaching the threshold at which the value becomes too low. Set the value a little higher than the threshold.
- When an ORB timeout value is set too low, the symptom is numerous CORBA NO_RESPONSE exceptions, which occur even for some valid requests. The value is likely to be too low if requests that should have been successful, for example, the server is not down, are being lost or refused.

Timeout adjustments: Do not adjust an ORB timeout value unless you have a problem. Configuring a value that is inappropriate for the environment can create a problem. An incorrect value can produce results worse than the original problem.

You can adjust timeout intervals for the product Java ORB through the following administrative settings:
- **Request timeout**, the number of seconds to wait before timing out on most pending ORB requests if the network fails
- **Locate request timeout**, the number of seconds to wait before timing out on a locate-request message

Read the information about Object Request Broker service settings in the *Setting up the application serving environment* PDF book.

Java packages containing the Object Request Broker service

The ORB service resides in the following Java packages:
- com.ibm.com.CORBA.*
- com.ibm.rmi.*
- com.ibm.ws.orb.*
- com.ibm.ws.orbimpl.*
- org.omg.CORBA.*
- javax.rmi.CORBA.*

JAR files that contain the previously mentioned packages include:
- *app_server_root*/java/jre/lib/ext/ibmorb.jar
- *app_server_root*/java/jre/lib/ext/iwsorbutil.jar
- *app_server_root*/lib/iwsorb.jar

Tools used with Object Request Broker

The tools used to compile Java remote interfaces to generate language bindings used by the ORB at runtime reside in the following Java packages:
- com.ibm.tools.rmic.*
- com.ibm.idl.*

The JAR file that contains the packages is *app_server_root*/java/lib/ibmtools.jar.

Object Request Broker properties

The ORB service requires a number of ORB properties for correct operation. It is not necessary for most users to modify these properties, and it is recommended that only your system administrator modify them when required.. Consult IBM Support personnel for assistance. The properties reside in the orb.properties file, located in *app_server_root*/java/jre/lib/orb.properties.

CORBA minor codes

The CORBA specification defines standard minor exception codes for use by the ORB when a system exception is thrown. In addition, the object management group (OMG) assigns each vendor a unique prefix value for use in vendor-proprietary minor exception codes. Minor code values assigned to IBM and used by the ORB in the WebSphere Application Server follow. The minor code value is in decimal and hexadecimal formats. The column labeled minor code reason gives a short description of the condition causing the exception. Currently there is no documentation for these errors beyond the minor code reason. If you require technical support from IBM, the minor code helps support engineers determine the source of the problem.

Table 48. Decimal minor exception codes 1229066320 to 1229066364

Decimal	Hexadecimal	Minor code reason

Table 48. Decimal minor exception codes 1229066320 to 1229066364 (continued)

1229066320	0x49421050	HTTP_READER_FAILURE
1229066321	0x49421051	COULD_NOT_INSTANTIATE_CLIENT_SSL_SOCKET_FACTORY
1229066322	0x49421052	COULD_NOT_INSTANTIATE_SERVER_SSL_SOCKET_FACTORY
1229066323	0x49421053	CREATE_LISTENER_FAILED_1
1229066324	0x49421054	CREATE_LISTENER_FAILED_2
1229066325	0x49421055	CREATE_LISTENER_FAILED_3
1229066326	0x49421056	CREATE_LISTENER_FAILED_4
1229066327	0x49421057	CREATE_LISTENER_FAILED_5
1229066328	0x49421058	INVALID_CONNECTION_TYPE
1229066329	0x49421059	HTTPINPUTSTREAM_NO_ACTIVEINPUTSTREAM
1229066330	0x4942105a	HTTPOUTPUTSTREAM_NO_OUTPUTSTREAM
1229066331	0x4942105b	CONNECTIONINTERCEPTOR_INVALID_CLASSNAME
1229066332	0x4942105c	NO_CONNECTIONDATA_IN_CONNECTIONDATACARRIER
1229066333	0x4942105d	CLIENT_CONNECTIONDATA_IS_INVALID_TYPE
1229066334	0x4942105e	SERVER_CONNECTIONDATA_IS_INVALID_TYPE
1229066335	0x4942105f	NO_OVERLAP_OF_ENABLED_AND_DESIRED_CIPHER_SUITES
1229066352	0x49421070	CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET
1229066353	0x49421071	GETCONNECTION_KEY_RETURNED_FALSE
1229066354	0x49421072	UNABLE_TO_CREATE_SSL_SOCKET
1229066355	0x49421073	SSLSERVERSOCKET_TARGET_SUPPORTS_LESS_THAN_1
1229066356	0x49421074	SSLSERVERSOCKET_TARGET_REQUIRES_LESS_THAN_1
1229066357	0x49421075	SSLSERVERSOCKET_TARGET_LESS_THAN_TARGET_REQUIRES
1229066358	0x49421076	UNABLE_TO_CREATE_SSL_SERVER_SOCKET
1229066359	0x49421077	CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_SERVER_SOCKET
1229066360	0x49421078	INVALID_SERVER_CONNECTION_DATA_TYPE
1229066361	0x49421079	GETSERVERCONNECTIONDATA_RETURNED_NULL
1229066362	0x4942107a	GET_SSL_SESSION_RETURNED_NULL
1229066363	0x4942107b	GLOBAL_ORB_EXISTS
1229066364	0x4942107c	CONNECT_TIME_OUT

Table 49. Decimal minor exception codes 1229123841 to 1229124249

Decimal	Hexadecimal	Minor code reason
1229123841	0x4942f101	DSIMETHOD_NOTCALLED
1229123842	0x4942f102	BAD_INV_PARAMS
1229123843	0x4942f103	BAD_INV_RESULT
1229123844	0x4942f104	BAD_INV_CTX
1229123845	0x4942f105	ORB_NOTREADY
1229123879	0x4942f127	PI_NOT_POST_INIT
1229123880	0x4942f128	INVALID_EXTENDED_PI_CALL
1229123969	0x4942f181	BAD_OPERATION_EXTRACT_SHORT
1229123970	0x4942f182	BAD_OPERATION_EXTRACT_LONG
1229123971	0x4942f183	BAD_OPERATION_EXTRACT_USHORT
1229123972	0x4942f184	BAD_OPERATION_EXTRACT_ULONG
1229123973	0x4942f185	BAD_OPERATION_EXTRACT_FLOAT
1229123974	0x4942f186	BAD_OPERATION_EXTRACT_DOUBLE
1229123975	0x4942f187	BAD_OPERATION_EXTRACT_LONGLONG
1229123976	0x4942f188	BAD_OPERATION_EXTRACT_ULONGLONG
1229123977	0x4942f189	BAD_OPERATION_EXTRACT_BOOLEAN
1229123978	0x4942f18a	BAD_OPERATION_EXTRACT_CHAR
1229123979	0x4942f18b	BAD_OPERATION_EXTRACT_OCTET
1229123980	0x4942f18c	BAD_OPERATION_EXTRACT_WCHAR

Table 49. Decimal minor exception codes 1229123841 to 1229124249 (continued)

1229123981	0x4942f18d	BAD_OPERATION_EXTRACT_STRING
1229123982	0x4942f18e	BAD_OPERATION_EXTRACT_WSTRING
1229123983	0x4942f18f	BAD_OPERATION_EXTRACT_ANY
1229123984	0x4942f190	BAD_OPERATION_INSERT_OBJECT_1
1229123985	0x4942f191	BAD_OPERATION_INSERT_OBJECT_2
1229123986	0x4942f192	BAD_OPERATION_EXTRACT_OBJECT_1
1229123987	0x4942f193	BAD_OPERATION_EXTRACT_OBJECT_2
1229123988	0x4942f194	BAD_OPERATION_EXTRACT_TYPECODE
1229123989	0x4942f195	BAD_OPERATION_EXTRACT_PRINCIPAL
1229123990	0x4942f196	BAD_OPERATION_EXTRACT_VALUE
1229123991	0x4942f197	BAD_OPERATION_GET_PRIMITIVE_TC_1
1229123992	0x4942f198	BAD_OPERATION_GET_PRIMITIVE_TC_2
1229123993	0x4942f199	BAD_OPERATION_INVOKE_NULL_PARAM_1
1229123994	0x4942f19a	BAD_OPERATION_INVOKE_NULL_PARAM_2
1229123995	0x4942f19b	BAD_OPERATION_INVOKE_DEFAULT_1
1229123996	0x4942f19c	BAD_OPERATION_INVOKE_DEFAULT_2
1229123997	0x4942f19d	BAD_OPERATION_UNKNOWN_BOOTSTRAP_METHOD
1229123998	0x4942f19e	BAD_OPERATION_EMPTY_ANY
1229123999	0x4942f19f	BAD_OPERATION_STUB_DISCONNECTED
1229124000	0x4942f1a0	BAD_OPERATION_TIE_DISCONNECTED
1229124001	0x4942f1a1	BAD_OPERATION_DELEGATE_DISCONNECTED
1229124097	0x4942f201	NULL_PARAM_1
1229124098	0x4942f202	NULL_PARAM_2
1229124099	0x4942f203	NULL_PARAM_3
1229124100	0x4942f204	NULL_PARAM_4
1229124101	0x4942f205	NULL_PARAM_5
1229124102	0x4942f206	NULL_PARAM_6
1229124103	0x4942f207	NULL_PARAM_7
1229124104	0x4942f208	NULL_PARAM_8
1229124105	0x4942f209	NULL_PARAM_9
1229124106	0x4942f20a	NULL_PARAM_10
1229124107	0x4942f20b	NULL_PARAM_11
1229124108	0x4942f20c	NULL_PARAM_12
1229124109	0x4942f20d	NULL_PARAM_13
1229124110	0x4942f20e	NULL_PARAM_14
1229124111	0x4942f20f	NULL_PARAM_15
1229124112	0x4942f210	NULL_PARAM_16
1229124113	0x4942f211	NULL_PARAM_17
1229124114	0x4942f212	NULL_PARAM_18
1229124115	0x4942f213	NULL_PARAM_19
1229124116	0x4942f214	NULL_PARAM_20
1229124117	0x4942f215	NULL_IOR_OBJECT
1229124118	0x4942f216	NULL_PI_NAME
1229124119	0x4942f217	NULL_SC_DATA
1229124126	0x4942f21e	BAD_SERVANT_TYPE
1229124127	0x4942f21f	BAD_EXCEPTION
1229124128	0x4942f220	BAD_MODIFIER_LIST
1229124129	0x4942f221	NULL_PROP_MGR
1229124130	0x4942f222	INVALID_PROPERTY
1229124131	0x4942f223	ORBINITREF_FORMAT
1229124132	0x4942f224	ORBINITREF_MISSING_OBJECTURL

Table 49. Decimal minor exception codes 1229123841 to 1229124249 (continued)

1229124133	0x4942f225	ORBDEFAULTINITREF_FORMAT
1229124134	0x4942f226	ORBDEFAULTINITREF_VALUE
1229124135	0x4942f227	OBJECTKEY_SERVERUUID_LENGTH
1229124136	0x4942f228	OBJECTKEY_SERVERUUID_NULL
1229124137	0x4942f229	BAD_REPOSITORY_ID
1229124138	0x4942f22a	BAD_PARAM_LOCAL_OBJECT
1229124139	0x4942f22b	NULL_OBJECT_IOR
1229124140	0x4942f22c	WRONG_ORB
1229124141	0x4942f22d	NULL_OBJECT_KEY
1229124142	0x4942f22e	NULL_OBJECT_URL
1229124143	0x4942f22f	NOT_A_NAMING_CONTEXT
1229124225	0x4942f281	TYPECODEIMPL_CTOR_MISUSE_1
1229124226	0x4942f282	TYPECODEIMPL_CTOR_MISUSE_2
1229124227	0x4942f283	TYPECODEIMPL_NULL_INDIRECTTYPE
1229124228	0x4942f284	TYPECODEIMPL_RECURSIVE_TYPECODES
1229124235	0x4942f28b	TYPECODEIMPL_KIND_INVALID_1
1229124236	0x4942f28c	TYPECODEIMPL_KIND_INVALID_2
1229124237	0x4942f28d	TYPECODEIMPL_NATIVE_1
1229124238	0x4942f28e	TYPECODEIMPL_NATIVE_2
1229124239	0x4942f28f	TYPECODEIMPL_NATIVE_3
1229124240	0x4942f290	TYPECODEIMPL_KIND_INDIRECT_1
1229124241	0x4942f291	TYPECODEIMPL_KIND_INDIRECT_2
1229124242	0x4942f292	TYPECODEIMPL_NULL_TYPECODE
1229124243	0x4942f293	TYPECODEIMPL_BODY_OF_TYPECODE
1229124244	0x4942f294	TYPECODEIMPL_KIND_RECURSIVE_1
1229124245	0x4942f295	TYPECODEIMPL_COMPLEX_DEFAULT_1
1229124246	0x4942f296	TYPECODEIMPL_COMPLEX_DEFAULT_2
1229124247	0x4942f297	TYPECODEIMPL_INDIRECTION
1229124248	0x4942f298	TYPECODEIMPL_SIMPLE_DEFAULT
1229124249	0x4942f299	TYPECODEIMPL_NOT_CDROS

Table 50. Decimal minor exception codes 1229124250 to 1229125764

Decimal	Hexadecimal	Minor code reason
1229124250	0x4942f29a	TYPECODEIMPL_NO_IMPLEMENT_1
1229124251	0x4942f29b	TYPECODEIMPL_NO_IMPLEMENT_2
1229124357	0x4942f305	CONN_PURGE_REBIND
1229124358	0x4942f306	CONN_PURGE_ABORT
1229124359	0x4942f307	CONN_NOT_ESTABLISH
1229124360	0x4942f308	CONN_CLOSE_REBIND
1229124368	0x4942f310	WRITE_ERROR_SEND
1229124376	0x4942f318	GET_PROPERTIES_ERROR
1229124384	0x4942f320	BOOTSTRAP_SERVER_NOT_AVAIL
1229124392	0x4942f328	INVOKE_ERROR
1229124481	0x4942f381	BAD_HEX_DIGIT
1229124482	0x4942f382	BAD_STRINGIFIED_IOR_LEN
1229124483	0x4942f383	BAD_STRINGIFIED_IOR
1229124485	0x4942f385	BAD_MODIFIER_1
1229124486	0x4942f386	BAD_MODIFIER_2
1229124488	0x4942f388	CODESET_INCOMPATIBLE
1229124490	0x4942f38a	LONG_DOUBLE_NOT_IMPLEMENTED_1
1229124491	0x4942f38b	LONG_DOUBLE_NOT_IMPLEMENTED_2

Table 50. Decimal minor exception codes 1229124250 to 1229125764 (continued)

1229124492	0x4942f38c	LONG_DOUBLE_NOT_IMPLEMENTED_3
1229124496	0x4942f390	COMPLEX_TYPES_NOT_IMPLEMENTED
1229124497	0x4942f391	VALUE_BOX_NOT_IMPLEMENTED
1229124498	0x4942f392	NULL_STRINGIFIED_IOR
1229124865	0x4942f501	TRANS_NS_CANNOT_CREATE_INITIAL_NC_SYS
1229124866	0x4942f502	TRANS_NS_CANNOT_CREATE_INITIAL_NC
1229124867	0x4942f503	GLOBAL_ORB_EXISTS
1229124868	0x4942f504	PLUGINS_ERROR
1229124869	0x4942f505	INCOMPATIBLE_JDK_VERSION
1229124993	0x4942f581	BAD_REPLYSTATUS
1229124994	0x4942f582	PEEKSTRING_FAILED
1229124995	0x4942f583	GET_LOCAL_HOST_FAILED
1229124996	0x4942f584	CREATE_LISTENER_FAILED
1229124997	0x4942f585	BAD_LOCATE_REQUEST_STATUS
1229124998	0x4942f586	STRINGIFY_WRITE_ERROR
1229125000	0x4942f588	BAD_GIOP_REQUEST_TYPE_1
1229125001	0x4942f589	BAD_GIOP_REQUEST_TYPE_2
1229125002	0x4942f58a	BAD_GIOP_REQUEST_TYPE_3
1229125003	0x4942f58b	BAD_GIOP_REQUEST_TYPE_4
1229125005	0x4942f58d	NULL_ORB_REFERENCE
1229125006	0x4942f58e	NULL_NAME_REFERENCE
1229125008	0x4942f590	ERROR_UNMARSHALING_USEREXC
1229125009	0x4942f591	SUBCONTRACTREGISTRY_ERROR
1229125010	0x4942f592	LOCATIONFORWARD_ERROR
1229125011	0x4942f593	BAD_READER_THREAD
1229125013	0x4942f595	BAD_REQUEST_ID
1229125014	0x4942f596	BAD_SYSTEMEXCEPTION
1229125015	0x4942f597	BAD_COMPLETION_STATUS
1229125016	0x4942f598	INITIAL_REF_ERROR
1229125017	0x4942f599	NO_CODEC_FACTORY
1229125018	0x4942f59a	BAD_SUBCONTRACT_ID
1229125019	0x4942f59b	BAD_SYSTEMEXCEPTION_2
1229125020	0x4942f59c	NOT_PRIMITIVE_TYPECODE
1229125021	0x4942f59d	BAD_SUBCONTRACT_ID_2
1229125022	0x4942f59e	BAD_SUBCONTRACT_TYPE
1229125023	0x4942f59f	NAMING_CTX_REBIND_ALREADY_BOUND
1229125024	0x4942f5a0	NAMING_CTX_REBINDCTX_ALREADY_BOUND
1229125025	0x4942f5a1	NAMING_CTX_BAD_BINDINGTYPE
1229125026	0x4942f5a2	NAMING_CTX_RESOLVE_CANNOT_NARROW_TO_CTX
1229125032	0x4942f5a8	TRANS_NC_BIND_ALREADY_BOUND
1229125033	0x4942f5a9	TRANS_NC_LIST_GOT_EXC
1229125034	0x4942f5aa	TRANS_NC_NEWCTX_GOT_EXC
1229125035	0x4942f5ab	TRANS_NC_DESTROY_GOT_EXC
1229125036	0x4942f5ac	TRANS_BI_DESTROY_GOT_EXC
1229125042	0x4942f5b2	INVALID_CHAR_CODESET_1
1229125043	0x4942f5b3	INVALID_CHAR_CODESET_2
1229125044	0x4942f5b4	INVALID_WCHAR_CODESET_1
1229125045	0x4942f5b5	INVALID_WCHAR_CODESET_2
1229125046	0x4942f5b6	GET_HOST_ADDR_FAILED
1229125047	0x4942f5b7	REACHED_UNREACHABLE_PATH
1229125048	0x4942f5b8	PROFILE_CLONE_FAILED

Table 50. Decimal minor exception codes 1229124250 to 1229125764 (continued)

1229125049	0x4942f5b9	INVALID_LOCATE_REQUEST_STATUS
1229125050	0x4942f5ba	NO_UNSAFE_CLASS
1229125051	0x4942f5bb	REQUEST_WITHOUT_CONNECTION_1
1229125052	0x4942f5bc	REQUEST_WITHOUT_CONNECTION_2
1229125053	0x4942f5bd	REQUEST_WITHOUT_CONNECTION_3
1229125054	0x4942f5be	REQUEST_WITHOUT_CONNECTION_4
1229125055	0x4942f5bf	REQUEST_WITHOUT_CONNECTION_5
1229125056	0x4942f5c0	NOT_USED_1
1229125057	0x4942f5c1	NOT_USED_2
1229125058	0x4942f5c2	NOT_USED_3
1229125059	0x4942f5c3	NOT_USED_4
1229125512	0x4942f788	BAD_CODE_SET
1229125520	0x4942f790	INV_RMI_STUB
1229125521	0x4942f791	INV_LOAD_STUB
1229125522	0x4942f792	INV_OBJ_IMPLEMENTATION
1229125523	0x4942f793	OBJECTKEY_NOMAGIC
1229125524	0x4942f794	OBJECTKEY_NOSCID
1229125525	0x4942f795	OBJECTKEY_NOSERVERID
1229125526	0x4942f796	OBJECTKEY_NOSERVERUUID
1229125527	0x4942f797	OBJECTKEY_SERVERUUIDKEY
1229125528	0x4942f798	OBJECTKEY_NOPOANAME
1229125529	0x4942f799	OBJECTKEY_SETSCID
1229125530	0x4942f79a	OBJECTKEY_SETSERVERID
1229125531	0x4942f79b	OBJECTKEY_NOUSERKEY
1229125532	0x4942f79c	OBJECTKEY_SETUSERKEY
1229125533	0x4942f79d	INVALID_INDEXED_PROFILE_1
1229125534	0x4942f79e	INVALID_INDEXED_PROFILE_2
1229125762	0x4942f882	UNSPECIFIED_MARSHAL_1
1229125763	0x4942f883	UNSPECIFIED_MARSHAL_2
1229125764	0x4942f884	UNSPECIFIED_MARSHAL_3

Table 51. Decimal minor exception codes 1299125765 to 1229125906

Decimal	Hexadecimal	Minor code reason
1229125765	0x4942f885	UNSPECIFIED_MARSHAL_4
1229125766	0x4942f886	UNSPECIFIED_MARSHAL_5
1229125767	0x4942f887	UNSPECIFIED_MARSHAL_6
1229125768	0x4942f888	UNSPECIFIED_MARSHAL_7
1229125769	0x4942f889	UNSPECIFIED_MARSHAL_8
1229125770	0x4942f88a	UNSPECIFIED_MARSHAL_9
1229125771	0x4942f88b	UNSPECIFIED_MARSHAL_10
1229125772	0x4942f88c	UNSPECIFIED_MARSHAL_11
1229125773	0x4942f88d	UNSPECIFIED_MARSHAL_12
1229125774	0x4942f88e	UNSPECIFIED_MARSHAL_13
1229125775	0x4942f88f	UNSPECIFIED_MARSHAL_14
1229125776	0x4942f890	UNSPECIFIED_MARSHAL_15
1229125777	0x4942f891	UNSPECIFIED_MARSHAL_16
1229125778	0x4942f892	UNSPECIFIED_MARSHAL_17
1229125779	0x4942f893	UNSPECIFIED_MARSHAL_18
1229125780	0x4942f894	UNSPECIFIED_MARSHAL_19
1229125781	0x4942f895	UNSPECIFIED_MARSHAL_20
1229125782	0x4942f896	UNSPECIFIED_MARSHAL_21

Table 51. Decimal minor exception codes 1299125765 to 1229125906 (continued)

1229125783	0x4942f897	UNSPECIFIED_MARSHAL_22
1229125784	0x4942f898	UNSPECIFIED_MARSHAL_23
1229125785	0x4942f899	UNSPECIFIED_MARSHAL_24
1229125786	0x4942f89a	UNSPECIFIED_MARSHAL_25
1229125787	0x4942f89b	UNSPECIFIED_MARSHAL_26
1229125788	0x4942f89c	UNSPECIFIED_MARSHAL_27
1229125789	0x4942f89d	UNSPECIFIED_MARSHAL_28
1229125790	0x4942f89e	UNSPECIFIED_MARSHAL_29
1229125791	0x4942f89f	UNSPECIFIED_MARSHAL_30
1229125792	0x4942f8a0	UNSPECIFIED_MARSHAL_31
1229125793	0x4942f8a1	UNSPECIFIED_MARSHAL_32
1229125794	0x4942f8a2	UNSPECIFIED_MARSHAL_33
1229125795	0x4942f8a3	UNSPECIFIED_MARSHAL_34
1229125796	0x4942f8a4	UNSPECIFIED_MARSHAL_35
1229125797	0x4942f8a5	UNSPECIFIED_MARSHAL_36
1229125798	0x4942f8a6	UNSPECIFIED_MARSHAL_37
1229125799	0x4942f8a7	UNSPECIFIED_MARSHAL_38
1229125800	0x4942f8a8	UNSPECIFIED_MARSHAL_39
1229125801	0x4942f8a9	UNSPECIFIED_MARSHAL_40
1229125802	0x4942f8aa	UNSPECIFIED_MARSHAL_41
1229125803	0x4942f8ab	UNSPECIFIED_MARSHAL_42
1229125804	0x4942f8ac	UNSPECIFIED_MARSHAL_43
1229125805	0x4942f8ad	UNSPECIFIED_MARSHAL_44
1229125806	0x4942f8ae	UNSPECIFIED_MARSHAL_45
1229125807	0x4942f8af	UNSPECIFIED_MARSHAL_46
1229125808	0x4942f8b0	UNSPECIFIED_MARSHAL_47
1229125809	0x4942f8b1	UNSPECIFIED_MARSHAL_48
1229125810	0x4942f8b2	UNSPECIFIED_MARSHAL_49
1229125811	0x4942f8b3	UNSPECIFIED_MARSHAL_50
1229125812	0x4942f8b4	UNSPECIFIED_MARSHAL_51
1229125813	0x4942f8b5	UNSPECIFIED_MARSHAL_52
1229125814	0x4942f8b6	UNSPECIFIED_MARSHAL_53
1229125815	0x4942f8b7	UNSPECIFIED_MARSHAL_54
1229125816	0x4942f8b8	UNSPECIFIED_MARSHAL_55
1229125817	0x4942f8b9	UNSPECIFIED_MARSHAL_56
1229125818	0x4942f8ba	UNSPECIFIED_MARSHAL_57
1229125819	0x4942f8bb	UNSPECIFIED_MARSHAL_58
1229125820	0x4942f8bc	UNSPECIFIED_MARSHAL_59
1229125821	0x4942f8bd	UNSPECIFIED_MARSHAL_60
1229125822	0x4942f8be	UNSPECIFIED_MARSHAL_61
1229125823	0x4942f8bf	UNSPECIFIED_MARSHAL_62
1229125824	0x4942f8c0	UNSPECIFIED_MARSHAL_63
1229125825	0x4942f8c1	UNSPECIFIED_MARSHAL_64
1229125826	0x4942f8c2	UNSPECIFIED_MARSHAL_65
1229125827	0x4942f8c3	UNSPECIFIED_MARSHAL_66
1229125828	0x4942f8c4	READ_OBJECT_EXCEPTION_2
1229125841	0x4942f8d1	UNSUPPORTED_IDLTYPE
1229125842	0x4942f8d2	DSI_RESULT_EXCEPTION
1229125844	0x4942f8d4	IIOPINPUTSTREAM_GROW
1229125847	0x4942f8d7	NO_CHAR_CONVERTER_1
1229125848	0x4942f8d8	NO_CHAR_CONVERTER_2

Table 51. Decimal minor exception codes 1299125765 to 1229125906 (continued)

1229125849	0x4942f8d9	CHARACTER_MALFORMED_1
1229125850	0x4942f8da	CHARACTER_MALFORMED_2
1229125851	0x4942f8db	CHARACTER_MALFORMED_3
1229125852	0x4942f8dc	CHARACTER_MALFORMED_4
1229125854	0x4942f8de	INCORRECT_CHUNK_LENGTH
1229125856	0x4942f8e0	CHUNK_OVERFLOW
1229125858	0x4942f8e2	CANNOT_GROW
1229125859	0x4942f8e3	CODESET_ALREADY_SET
1229125860	0x4942f8e4	REQUEST_CANCELLED
1229125861	0x4942f8e5	WRITE_TO_STREAM_1
1229125862	0x4942f8e6	WRITE_TO_STREAM_2
1229125863	0x4942f8e7	WRITE_TO_STREAM_3
1229125864	0x4942f8e8	WRITE_TO_STREAM_4
1229125865	0x4942f8e9	PROXY_MARSHAL_FAILURE
1229125866	0x4942f8ea	PROXY_UNMARSHAL_FAILURE
1229125867	0x4942f8eb	INVALID_START_VALUE
1229125868	0x4942f8ec	INVALID_CHUNK_STATE
1229125869	0x4942f8ed	NULL_CODEBASE_IOR
1229125889	0x4942f901	DSI_NOT_IMPLEMENTED
1229125890	0x4942f902	GETINTERFACE_NOT_IMPLEMENTED
1229125891	0x4942f903	SEND_DEFERRED_NOTIMPLEMENTED
1229125893	0x4942f905	ARGUMENTS_NOTIMPLEMENTED
1229125894	0x4942f906	RESULT_NOTIMPLEMENTED
1229125895	0x4942f907	EXCEPTIONS_NOTIMPLEMENTED
1229125896	0x4942f908	CONTEXTLIST_NOTIMPLEMENTED
1229125902	0x4942f90e	CREATE_OBJ_REF_BYTE_NOTIMPLEMENTED
1229125903	0x4942f90f	CREATE_OBJ_REF_IOR_NOTIMPLEMENTED
1229125904	0x4942f910	GET_KEY_NOTIMPLEMENTED
1229125905	0x4942f911	GET_IMPL_ID_NOTIMPLEMENTED
1229125906	0x4942f912	GET_SERVANT_NOTIMPLEMENTED

Table 52. Decimal minor exception codes 1229125907 to 1229126567

Decimal	Hexadecimal	Minor code reason
1229125907	0x4942f913	SET_ORB_NOTIMPLEMENTED
1229125908	0x4942f914	SET_ID_NOTIMPLEMENTED
1229125909	0x4942f915	GET_CLIENT_SUBCONTRACT_NOTIMPLEMENTED
1229125913	0x4942f919	CONTEXTIMPL_NOTIMPLEMENTED
1229125914	0x4942f91a	CONTEXT_NAME_NOTIMPLEMENTED
1229125915	0x4942f91b	PARENT_NOTIMPLEMENTED
1229125916	0x4942f91c	CREATE_CHILD_NOTIMPLEMENTED
1229125917	0x4942f91d	SET_ONE_VALUE_NOTIMPLEMENTED
1229125918	0x4942f91e	SET_VALUES_NOTIMPLEMENTED
1229125919	0x4942f91f	DELETE_VALUES_NOTIMPLEMENTED
1229125920	0x4942f920	GET_VALUES_NOTIMPLEMENTED
1229125922	0x4942f922	GET_CURRENT_NOTIMPLEMENTED_1
1229125923	0x4942f923	GET_CURRENT_NOTIMPLEMENTED_2
1229125924	0x4942f924	CREATE_OPERATION_LIST_NOTIMPLEMENTED_1
1229125925	0x4942f925	CREATE_OPERATION_LIST_NOTIMPLEMENTED_2
1229125926	0x4942f926	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_1
1229125927	0x4942f927	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_2
1229125928	0x4942f928	SHUTDOWN_NOTIMPLEMENTED

Table 52. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229125929	0x4942f929	WORK_PENDING_NOTIMPLEMENTED
1229125930	0x4942f92a	PERFORM_WORK_NOTIMPLEMENTED
1229125931	0x4942f92b	COPY_TK_ABSTRACT_NOTIMPLEMENTED
1229125932	0x4942f92c	PI_CLIENT_GET_POLICY_NOTIMPLEMENTED
1229125933	0x4942f92d	PI_SERVER_GET_POLICY_NOTIMPLEMENTED
1229125934	0x4942f92e	ADDRESSING_MODE_NOTIMPLEMENTED_1
1229125935	0x4942f92f	ADDRESSING_MODE_NOTIMPLEMENTED_2
1229125936	0x4942f930	SET_OBJECT_RESOLVER_NOTIMPLEMENTED
1229125937	0x4942f931	DISCONNECTED_SERVANT_1
1229125938	0x4942f932	DISCONNECTED_SERVANT_2
1229125939	0x4942f933	DISCONNECTED_SERVANT_3
1229125940	0x4942f934	DISCONNECTED_SERVANT_4
1229125941	0x4942f935	DISCONNECTED_SERVANT_5
1229125942	0x4942f936	DISCONNECTED_SERVANT_6
1229125943	0x4942f937	DISCONNECTED_SERVANT_7
1229125944	0x4942f938	GET_INTERFACE_DEF_NOT_IMPLEMENTED
1229126017	0x4942f981	MARSHAL_NO_MEMORY_1
1229126018	0x4942f982	MARSHAL_NO_MEMORY_2
1229126019	0x4942f983	MARSHAL_NO_MEMORY_3
1229126020	0x4942f984	MARSHAL_NO_MEMORY_4
1229126021	0x4942f985	MARSHAL_NO_MEMORY_5
1229126022	0x4942f986	MARSHAL_NO_MEMORY_6
1229126023	0x4942f987	MARSHAL_NO_MEMORY_7
1229126024	0x4942f988	MARSHAL_NO_MEMORY_8
1229126025	0x4942f989	MARSHAL_NO_MEMORY_9
1229126026	0x4942f98a	MARSHAL_NO_MEMORY_10
1229126027	0x4942f98b	MARSHAL_NO_MEMORY_11
1229126028	0x4942f98c	MARSHAL_NO_MEMORY_12
1229126029	0x4942f98d	MARSHAL_NO_MEMORY_13
1229126030	0x4942f98e	MARSHAL_NO_MEMORY_14
1229126031	0x4942f98f	MARSHAL_NO_MEMORY_15
1229126032	0x4942f990	MARSHAL_NO_MEMORY_16
1229126033	0x4942f991	MARSHAL_NO_MEMORY_17
1229126034	0x4942f992	MARSHAL_NO_MEMORY_18
1229126035	0x4942f993	MARSHAL_NO_MEMORY_19
1229126036	0x4942f994	MARSHAL_NO_MEMORY_20
1229126037	0x4942f995	MARSHAL_NO_MEMORY_21
1229126038	0x4942f996	MARSHAL_NO_MEMORY_22
1229126039	0x4942f997	MARSHAL_NO_MEMORY_23
1229126040	0x4942f998	MARSHAL_NO_MEMORY_24
1229126041	0x4942f999	MARSHAL_NO_MEMORY_25
1229126042	0x4942f99a	MARSHAL_NO_MEMORY_26
1229126043	0x4942f99b	MARSHAL_NO_MEMORY_27
1229126044	0x4942f99c	MARSHAL_NO_MEMORY_28
1229126045	0x4942f99d	MARSHAL_NO_MEMORY_29
1229126046	0x4942f99e	MARSHAL_NO_MEMORY_30
1229126047	0x4942f99f	MARSHAL_NO_MEMORY_31
1229126401	0x4942fb01	RESPONSE_TIMED_OUT
1229126402	0x4942fb02	FRAGMENT_TIMED_OUT
1229126529	0x4942fb81	NO_SERVER_SC_IN_DISPATCH
1229126530	0x4942fb82	NO_SERVER_SC_IN_LOOKUP

Table 52. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229126531	0x4942fb83	NO_SERVER_SC_IN_CREATE_DEFAULT_SERVER
1229126532	0x4942fb84	NO_SERVER_SC_IN_SETUP
1229126533	0x4942fb85	NO_SERVER_SC_IN_LOCATE
1229126534	0x4942fb86	NO_SERVER_SC_IN_DISCONNECT
1229126539	0x4942fb8b	ORB_CONNECT_ERROR_1
1229126540	0x4942fb8c	ORB_CONNECT_ERROR_2
1229126541	0x4942fb8d	ORB_CONNECT_ERROR_3
1229126542	0x4942fb8e	ORB_CONNECT_ERROR_4
1229126543	0x4942fb8f	ORB_CONNECT_ERROR_5
1229126544	0x4942fb90	ORB_CONNECT_ERROR_6
1229126545	0x4942fb91	ORB_CONNECT_ERROR_7
1229126546	0x4942fb92	ORB_CONNECT_ERROR_8
1229126547	0x4942fb93	ORB_CONNECT_ERROR_9
1229126548	0x4942fb94	ORB_REGISTER_1
1229126549	0x4942fb95	ORB_REGISTER_2
1229126553	0x4942fb99	ORB_REGISTER_LOCAL_1
1229126554	0x4942fb9a	ORB_REGISTER_LOCAL_2
1229126555	0x4942fb9b	LOCAL_SERVANT_LOOKUP
1229126556	0x4942fb9c	POA_LOOKUP_ERROR
1229126557	0x4942fb9d	POA_INACTIVE
1229126558	0x4942fb9e	POA_NO_SERVANT_MANAGER
1229126559	0x4942fb9f	POA_NO_DEFAULT_SERVANT
1229126560	0x4942fba0	POA_WRONG_POLICY
1229126561	0x4942fba1	FINDPOA_ERROR
1229126562	0x4942fba2	ADAPTER_ACTIVATOR_EXCEPTION
1229126563	0x4942fba3	POA_SERVANT_ACTIVATOR_LOOKUP_FAILED
1229126564	0x4942fba4	POA_BAD_SERVANT_MANAGER
1229126565	0x4942fba5	POA_SERVANT_LOCATOR_LOOKUP_FAILED
1229126566	0x4942fba6	POA_UNKNOWN_POLICY
1229126567	0x4942fba7	POA_NOT_FOUND

Table 53. Decimal minor exception codes 1229126568 to 1330446377

Decimal	Hexadecimal	Minor code reason
1229126568	0x4942fba8	SERVANT_LOOKUP
1229126569	0x4942fba9	SERVANT_IS_ACTIVE
1229126570	0x4942fbaa	SERVANT_DISPATCH
1229126571	0x4942fbab	WRONG_CLIENTSC
1229126572	0x4942fbac	WRONG_SERVERSC
1229126573	0x4942fbad	SERVANT_IS_NOT_ACTIVE
1229126574	0x4942fbae	POA_WRONG_POLICY_1
1229126575	0x4942fbaf	POA_NOCONTEXT_1
1229126576	0x4942fbb0	POA_NOCONTEXT_2
1229126577	0x4942fbb1	POA_INVALID_NAME_1
1229126578	0x4942fbb2	POA_INVALID_NAME_2
1229126579	0x4942fbb3	POA_INVALID_NAME_3
1229126580	0x4942fbb4	POA_NOCONTEXT_FOR_PREINVOKE
1229126581	0x4942fbb5	POA_NOCONTEXT_FOR_CHECKING_PREINVOKE
1229126582	0x4942fbb6	POA_NOCONTEXT_FOR_POSTINVOKE
1229126657	0x4942fc01	LOCATE_UNKNOWN_OBJECT
1229126658	0x4942fc02	BAD_SERVER_ID_1
1229126659	0x4942fc03	BAD_SERVER_ID_2

Table 53. Decimal minor exception codes 1229126568 to 1330446377 (continued)

1229126660	0x4942fc04	BAD_IMPLID
1229126665	0x4942fc09	BAD_SKELETON_1
1229126666	0x4942fc0a	BAD_SKELETON_2
1229126673	0x4942fc11	SERVANT_NOT_FOUND_1
1229126674	0x4942fc12	SERVANT_NOT_FOUND_2
1229126675	0x4942fc13	SERVANT_NOT_FOUND_3
1229126676	0x4942fc14	SERVANT_NOT_FOUND_4
1229126677	0x4942fc15	SERVANT_NOT_FOUND_5
1229126678	0x4942fc16	SERVANT_NOT_FOUND_6
1229126679	0x4942fc17	SERVANT_NOT_FOUND_7
1229126687	0x4942fc1f	SERVANT_DISCONNECTED_1
1229126688	0x4942fc20	SERVANT_DISCONNECTED_2
1229126689	0x4942fc21	NULL_SERVANT
1229126690	0x4942fc22	ADAPTER_ACTIVATOR_FAILED
1229126692	0x4942fc24	ORB_DESTROYED
1229126693	0x4942fc25	DYNANY_DESTROYED
1229127170	0x4942fe02	CONNECT_FAILURE_1
1229127171	0x4942fe03	CONNECT_FAILURE_2
1229127172	0x4942fe04	CONNECT_FAILURE_3
1229127173	0x4942fe05	CONNECT_FAILURE_4
1229127297	0x4942fe81	UNKNOWN_CORBA_EXC
1229127298	0x4942fe82	RUNTIMEEXCEPTION
1229127299	0x4942fe83	UNKNOWN_SERVER_ERROR
1229127300	0x4942fe84	UNKNOWN_DSI_SYSEX
1229127301	0x4942fe85	UNEXPECTED_CHECKED_EXCEPTION
1229127302	0x4942fe86	UNKNOWN_CREATE_EXCEPTION_RESPONSE
1229127312	0x4942fe90	UNKNOWN_PI_EXC
1229127313	0x4942fe91	UNKNOWN_PI_EXC_2
1229127314	0x4942fe92	PI_ARGS_FAILURE
1229127315	0x4942fe93	PI_EXCEPTS_FAILURE
1229127316	0x4942fe94	PI_CONTEXTS_FAILURE
1229127317	0x4942fe95	PI_OP_CONTEXT_FAILURE
1229127326	0x4942fe9e	USER_DEFINED_ERROR
1229127327	0x4942fe9f	UNKNOWN_RUNTIME_IN_BOOTSTRAP
1229127328	0x4942fea0	UNKNOWN_THROWABLE_IN_BOOTSTRAP
1229127329	0x4942fea1	UNKNOWN_RUNTIME_IN_INSAGENT
1229127330	0x4942fea2	UNKNOWN_THROWABLE_IN_INSAGENT
1229127331	0x4942fea3	UNEXPECTED_IN_PROCESSING_CLIENTSIDE_INTERCEPTOR
1229127332	0x4942fea4	UNEXPECTED_IN_PROCESSING_SERVERSIDE_INTERCEPTOR
1229127333	0x4942fea5	UNEXPECTED_PI_LOCAL_REQUEST
1229127334	0x4942fea6	UNEXPECTED_PI_LOCAL_RESPONSE
1330446336	0x4f4d0000	OMGVMCID
1330446337	0x4f4d0001	FAILURE_TO_REGISTER_OR_LOOKUP_VALUE_FACTORY
1330446338	0x4f4d0002	RID_ALREADY_DEFINED_IN_IFR
1330446339	0x4f4d0003	IN_INVOCATION_CONTEXT
1330446340	0x4f4d0004	ORB_SHUTDOWN
1330446341	0x4f4d0005	NAME_CLASH_IN_INHERITED_CONTEXT
1330446342	0x4f4d0006	SERVANT_MANAGER_EXISTS
1330446343	0x4f4d0007	INS_BAD_SCHEME_NAME
1330446344	0x4f4d0008	INS_BAD_ADDRESS
1330446345	0x4f4d0009	INS_BAD_SCHEME_SPECIFIC_PART

Table 53. Decimal minor exception codes 1229126568 to 1330446377 (continued)

1330446346	0x4f4d000a	INS_OTHER
1330446348	0x4f4d000c	POLICY_FACTORY_EXISTS
1330446350	0x4f4d000e	INVALID_PI_CALL
1330446351	0x4f4d000f	SERVICE_CONTEXT_ID_EXISTS
1330446353	0x4f4d0011	POA_DESTROYED
1330446359	0x4f4d0017	NO_TRANSMISSION_CODE
1330446362	0x4f4d001a	INVALID_SERVICE_CONTEXT
1330446363	0x4f4d001b	NULL_OBJECT_ON_REGISTER
1330446364	0x4f4d001c	INVALID_COMPONENT_ID
1330446365	0x4f4d001d	INVALID_IOR_PROFILE
1330446375	0x4f4d0027	INVALID_STREAM_FORMAT_1
1330446376	0x4f4d0028	NOT_VALUE_OUTPUT_STREAM
1330446377	0x4f4d0029	NOT_VALUE_INPUT_STREAM

If none of these steps fixes your problem, 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 do not find your problem listed there, contact IBM Support.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page. You should also refer to this page before opening a PMR because it contains documents that can save you time gathering information needed to resolve a problem.

Chapter 19. Transactions

Using the transaction service

These topics provide information about using transactions with WebSphere applications

WebSphere applications can use transactions to coordinate multiple updates to resources as atomic units (as indivisible units of work) such that all or none of the updates are made permanent.

In WebSphere Application Server, transactions are handled by three main components:
- A transaction manager. The transaction manager supports the enlistment of recoverable XAResources and ensures that each such resource is driven to a consistent outcome either at the end of a transaction or after a failure and restart of the application server.
- A container in which the J2EE application runs. The container manages the enlistment of XAResources on behalf of the application when the application performs updates to transactional resource managers (for example, databases). Optionally, the container can control the demarcation of transactions for enterprise beans configured for container-managed transactions.
- An application programming interface (UserTransaction) that is available to bean-managed enterprise beans and servlets. This allows such application components to control the demarcation of their own transactions.

For more information about using transactions with WebSphere applications, see the following topics:
- “Transaction support in WebSphere Application Server”
- Developing components to use transactions
- “Configuring transaction properties for an application server” on page 1861
- “Use of local transactions” on page 1856
- “Managing active and prepared transactions” on page 1871
- “Managing transaction logging for optimum server availability” on page 1872
- “Interoperating transactionally between application servers” on page 1875
- “Configuring an intermediary node for Web services transactions” on page 1876
- “Enabling WebSphere Application Server to use an intermediary node for Web services transactions” on page 1876
- “Configuring a server to use business activity support” on page 1877
- Creating an application that exploits the business activity support
- “The business activity API” on page 1858
- Troubleshooting transactions
- “Transaction service exceptions” on page 1860
- “UserTransaction interface - methods available” on page 1861
- Using one-phase and two-phase commit resources in the same transaction
-
-
- Using the ActivitySession service

Transaction support in WebSphere Application Server

This topic provides conceptual information about the support for transactions provided by the Transaction Service of WebSphere Application Server.

A transaction is unit of activity within which multiple updates to resources can be made atomic (as an indivisible unit of work) such that all or none of the updates are made permanent. For example, multiple SQL statements to a relational database are committed atomically by the database during the processing of an SQL COMMIT statement. In this case, the transaction is contained entirely within the database manager and can be thought of as a *resource manager local transaction (RMLT)*. In some contexts, a transaction is referred to as a *logical unit of work* (LUW). If a transaction involves multiple resource managers, for example multiple database managers, then an external transaction manager is required to

coordinate the individual resource managers. A transaction that spans multiple resource managers are referred to as a *global transaction*. WebSphere Application Server is a transaction manager that can coordinate global transactions, be a participant in a received global transaction and also provides an environment in which resource manager local transactions can run.

The way that applications use transactions depends on the type of application component, as follows:
- A session bean can either use container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (component-managed transactions where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) and application client components use component-managed transactions.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service (OTS) protocol or Web Service Atomic Transaction (WS-AtomicTransaction) protocol. WebSphere Application Server also participates in transactions imported through J2EE Connector 1.5 resource adapters. You can also configure WebSphere applications to interact with databases, JMS queues, and JCA connectors through their *local transaction* support, when you do not require distributed transaction coordination.

Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

Under normal circumstances you cannot mix one-phase commit capable resources and two-phase commit capable resources in the same global transaction, because one-phase commit resources cannot support the prepare phase of two-phase commit. There are some special circumstances where it is possible to include mixed-capability resources in the same global transaction:
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction and where all the two-phase commit resource-providers that participate in the transaction are used in a read-only fashion. In this case, the two-phase commit resources all vote read-only during the prepare phase of two-phase commit. Because the one-phase commit resource provider is the only provider to actually perform any updates, the one-phase commit resource does not need to be prepared.
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction with one of more two-phase commit resource providers and where *last participant support* is enabled. Last participant support enables the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction. For more information about last participant support, see Using one-phase and two-phase commit resources in the same transaction.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the J2EE programming model. EJBs can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager for the period of a client-scoped ActivitySession, rather than only the duration of an EJB method, and have the resource manager's local transaction outcome directed by the ActivitySession. For more information about ActivitySessions, see Using the ActivitySession service.

Resource manager local transaction (RMLT)

A resource manager local transaction (RMLT) is a resource manager's view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:
- Enterprise Information Systems that are accessed through a resource adapter, as described in the J2EE Connector Architecture 1.0.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. J2EE connector resource adapters that include support for local transactions provide a LocalTransaction interface to enable applications to request that the resource adapter commit or rollback RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in WebSphere Application Server by a local transaction containment (LTC).

Global transactions

If an application uses two or more resources, then an external transaction manager is needed to coordinate the updates to both resource managers in a *global tansaction*.

Global transaction support is available to web and enterprise bean J2EE components and, with some limitation, to application client components. Enterprise bean components can be subdivided into beans that exploit container-managed transactions (CMT) or bean-managed transactions (BMT).

BMT enterprise beans, application client components, and web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. The UserTransaction interface can be obtained by a JNDI lookup of `java:comp/UserTransaction` or from the SessionContext object using the getUserTransaction method..

The UserTransaction is not available to the following components:
- CMT enterprise beans. Any attempt by such beans to obtain the interface results in an exception in accordance with the EJB specification.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface, use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location appropriate for the EJB version.

Before the EJB 1.1 specification, the JNDI location of the UserTransaction interface was not specified. Each EJB container implementor defined it in an implementation-specific manner. Earlier versions of WebSphere Application Server, up to and including Version 3.5.x (without EJB 1.1), bind the UserTransaction interface to a JNDI location of jta/usertransaction. WebSphere Application Server Version 4, and later releases, bind the UserTransaction interface at the location defined by EJB 1.1, which is java:comp/UserTransaction. WebSphere Application Server, from Version 5 no longer provides the jta/usertransaction binding within Web and EJB containers to applications at a J2EE level of 1.3 or later. For example, from EJB 2.0 applications can use only the java:comp/UserTransaction location.

A web component or enterprise bean (CMT or BMT) can get the ExtendedJTATransaction interface through a lookup of `java:comp/websphere/ExtendedJTATransaction`. This interface provides access to the transaction identity and a mechanism to receive notification of transaction completion.

Local transaction containment (LTC)

A *local transaction containment (LTC)* is used to define the application server behavior in an unspecified transaction context.

(Unspecified transaction context is defined in the Enterprise JavaBeans 2.0 (or later) specification; for example, at http://java.sun.com/products/ejb/2.0.html.)

A LTC is a bounded unit-of-work scope within which zero, one, or more resource manager local transactions (RMLTs) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. An LTC is local to a bean instance; it is not shared across beans even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on a J2EE component (such as an enterprise bean or servlet) whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary; for example at the end of the method dispatch. There is no programmatic interface to the LTC support; rather LTCs are managed exclusively by the container and configured by the application deployer through transaction attributes in the application deployment descriptor.

A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC for J2EE components at J2EE 1.3 or later. The only exceptions to this are as follows:
- Where application component dispatch occurs without container interposition; for example, for a stateless session bean `create` or a servlet-initiated thread.
- J2EE 1.2 web components.
- J2EE 1.2 BMT enterprise beans.

A local transaction containment can be scoped to an ActivitySession context that lives longer than the enterprise bean method in which it is started, as described in ActivitySessions and transaction contexts.

Local and global transaction considerations

Applications use resources, such as JDBC data sources or connection factories, that are configured through the Resources view of the WebSphere Application Server Administrative Console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider. For example, most JDBC providers can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLTs), but an XA data source can support two-phase commit coordination, as well as local transactions.

If an application uses two or more resource providers that support only RMLTs, then atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application should use resources that support XA coordination and should access them within a global transaction.

If an application uses only one RMLT, the atomic behavior can be guaranteed by the resource manager, which can be accessed under a local transaction containment context.

An application can also access a single resource manager under a global transaction context, even if that resource manager does not support the XA coordination. An application can do this, because WebSphere Application Server performs an “only resource optimization” and interacts with the resource manager under a RMLT. Within a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but never both. An instance of an enterprise bean can change from running under one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Client support for transactions

This topic describes the support of application clients for the use of transactions.

Application clients running in a J2EE client container can explicitly demarcate transaction boundaries as described in Using component-managed transactions. Application clients cannot perform, directly within the client container, transactional work in the context of any global transaction that they start, because the client container is not a recoverable process.

Application clients can make requests to remote objects, such as enterprise beans, within the context of a client-initiated transaction. Any transactional work performed in a remote, recoverable server process is coordinated as part of the client-initiated transaction. The transaction coordinator is created on the first server process to which the client-initiated transaction is propagated.

A client can begin a transaction then, for example, access a JDBC data source directly in the client process. In such cases, any work performed through the JDBC provider is not coordinated as part of the global transaction. Instead, the work runs under a resource manager local transaction. The client container process is non-recoverable and contains no transaction coordinator with which a resource manager can be enlisted.

A client can begin a transaction then call a remote application component, such as an enterprise bean. In such cases, the client-initiated transaction context is implicitly propagated to the remote application server where a transaction coordinator is created. Any resource managers accessed on the recoverable application server (or any other application server hosting application components invoked by the client) are enlisted in the global transaction.

Client application components need to be aware that locally-accessed resource managers are not coordinated by client-initiated transactions. Client applications acknowledge this through a deployment option that enables access to the UserTransaction interface in the client container. By default, access to the UserTransaction interface in the client container is not enabled. To enable UserTransaction demarcation for an application client component, set the **Allow JTA Demarcation** extension property in the client deployment descriptor. For information about editing the client deployment descriptor, refer to the Application Server Toolkit information, in the navigation pane of this infocenter.

Transaction compensation and business activity support

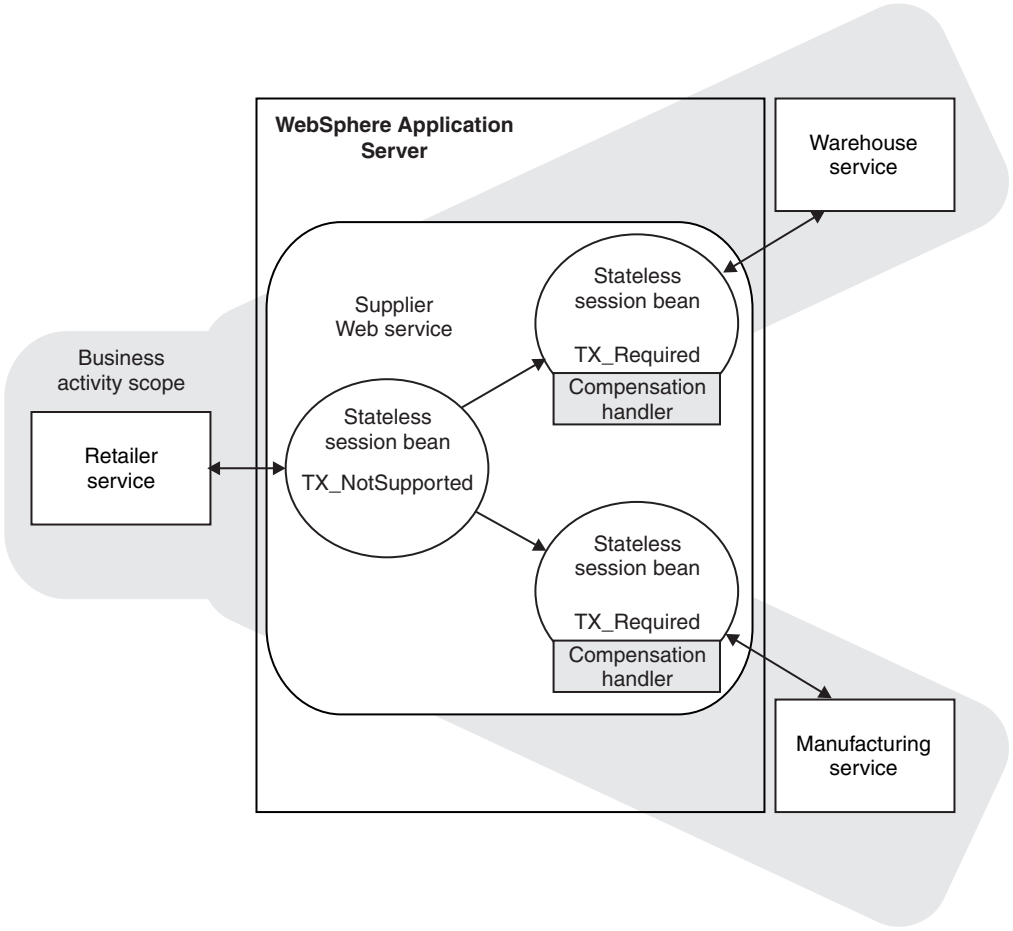
A *business activity* is a collection of tasks that are linked together so that they have an agreed outcome. Unlike atomic transactions, activities such as sending an e-mail can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error. The WebSphere Application Server business activity support provides this compensation ability through *business activity scopes*.

When to use business activity support

Use the business activity support when you have an application that requires compensation. An application requires compensation if its operations cannot be atomically rolled back. Typically, this scenario is due to one of the following reasons:

- The application uses multiple non-extended-architecture (XA) resources.
- The application uses more than one atomic transaction, for example, enterprise beans that have **Requires new** as the setting for the **Transaction** field in the container transaction deployment descriptor.
- The application does not run under a global transaction.

The following diagram shows a simple Web service application that uses the business activity support. The Retailer, Warehouse and Manufacturing services are running in non-WebSphere Application Server environments. The Retailer service calls the Supplier service, running on WebSphere Application Server, which delegates tasks to the Warehouse and Manufacturing services. The implementation of the Supplier service contains a stateless session bean, which calls other stateless session beans that are associated with the Warehouse and Manufacturing services, and that perform compensatable work. These other session beans each have a *compensation handler*, a piece of logic that is associated with an application component at run time, and performs compensation activity such as resending an e-mail.



Application design considerations

Business activity contexts are propagated with application messages, and can therefore be distributed between application components that are not co-located in the same server. Unlike atomic transaction contexts, business activity contexts are propagated on both synchronous (blocking) call-response messages and asynchronous one-way messages. An application component that runs under a business activity scope is responsible for ensuring that any asynchronous work it initiates is complete before the component's own processing is complete. An application that initiates asynchronous work using a fire-and-forget message pattern must not use business activity scopes, because such applications have no means of detecting whether this asynchronous processing has completed.

Only enterprise beans that have container-managed transactions can use the business activity functionality. Enterprise beans that exploit business activity scopes can offer Web service interfaces, but can also offer standard enterprise bean local or remote Java interfaces. Business activity context is propagated in Web service messages using a standard, interoperable Web Services Business Activity `CoordinationContext` element. WebSphere Application Server can also propagate business activity context on RMI calls to enterprise beans when Web services are not being used, but this form of the context is not interoperable with non-WebSphere Application Server environments. You might want to use this homogeneous scenario if you require compensation for an application that is internal to your business. If you want to use business activity compensation in a heterogeneous environment, expose your application components as Web services.

Business activity contexts can be propagated across firewalls and outside the WebSphere Application Server domain. The topology that you use to achieve this propagation can affect the high availability and affinity behavior of the business activity transaction.

Application development and deployment considerations

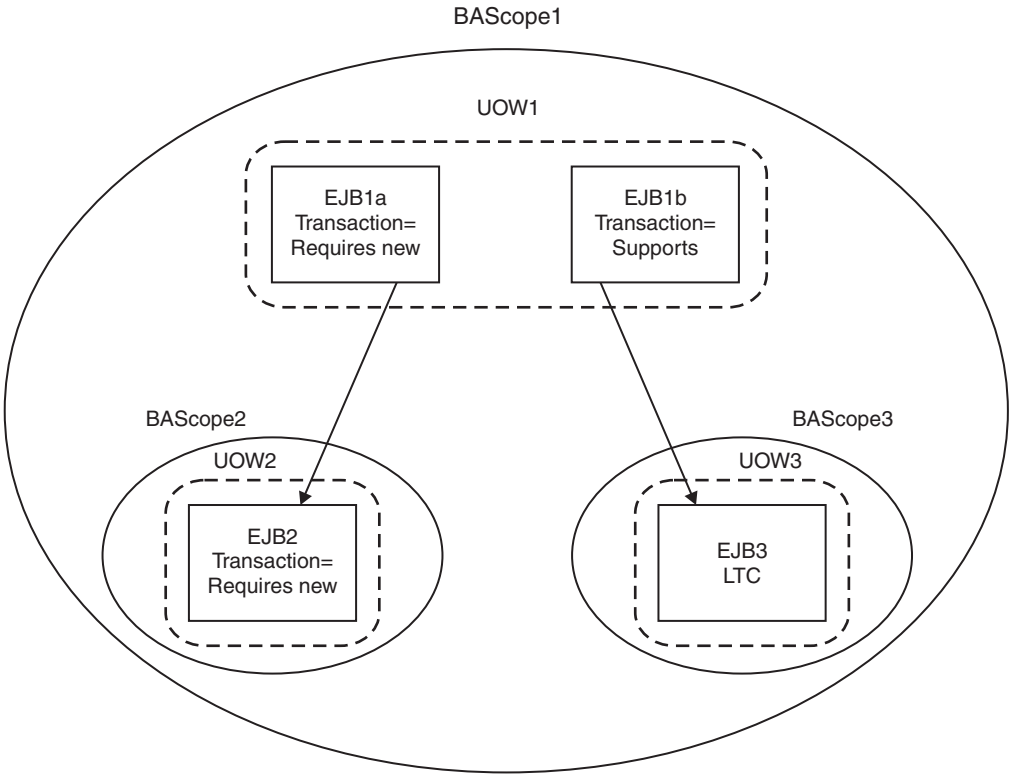
WebSphere Application Server provides a programming model for creating business activity scopes, and for associating compensation handlers with those business activity scopes. WebSphere Application Server also provides an application programming interface to specify compensation data, and check or alter the status of a business activity. To use the business activity support you must set certain application deployment descriptors appropriately, provide a compensation handler class if required, and enable business activity support on any servers that run the application.

Note: Applications can exploit the business activity support only if you deploy them to a WebSphere Application Server Version 6.1 server. Applications cannot use the business activity support if you deploy them to a cluster that includes WebSphere Application Server Version 6.0.x servers.

Business activity scopes

The scope of a business activity is that of a core WebSphere Application Server unit of work: a global transaction, an activity session, or local transaction containment (LTC). A business activity scope is not a new unit of work (UOW); it is an attribute of an existing core UOW. Therefore, a one-to-one relationship exists between a business activity scope and a UOW.

Any core UOW can have a business activity scope associated with it. If a component running under a UOW that is associated with a business activity scope calls another component, that request propagates the business activity scope; any work done by the new component is associated with the same business activity scope as the calling component. The called component can create a new UOW, for example if an enterprise bean has a **Transaction** setting of **Requires new**, or runs under the same UOW as the calling component. If a new UOW is started then a new business activity scope is created and associated with the new UOW. The newly created business activity scope is a child of the business activity scope associated with the calling UOW. In the following diagram, EJB1a running under UOW1 calls two components: EJB1b that also runs under UOW1, and EJB2 that creates a new UOW, UOW2. The enterprise bean EJB1b, calls another enterprise bean, EJB3, which creates another new UOW, UOW3. Because each new UOW is created by a calling component whose UOW already has an association with business activity scope BAScope1, the newly created UOWs are associated with new inner business activity scopes, BAScope2 and BAScope3.



Inner business activity scopes must complete before the outer business activity scope completes. Inner business activity scopes, for example BAScope2, have an association with the outer business activity scope, in this case BAScope1. Each business activity scope is directed to close if its associated UOW completes successfully, or to compensate if its associated UOW fails. If BAScope2 completes successfully, any active compensation handlers that are owned by BAScope2 are moved to BAScope1, and are directed in the same way as the completion direction of BAScope1: either compensate or close. If BAScope2 fails, the active compensation handlers are compensated automatically, and nothing is moved to the outer BAScope1. When an inner business activity scope fails, as a result of its associated UOW failing, an application server exception is thrown to the to calling application component, running in the outer UOW.

For example, if the inner UOW fails it might throw a TransactionRolledBackException exception. If the calling application can handle the exception, for example by retrying the called component or by calling another component, then the calling UOW, and its associated business activity scope, can complete successfully even though the inner business activity scope failed. If the application design requires the calling UOW to fail, and for its associated business activity scope to be compensated, then the calling application component must cause its UOW to fail, for example by allowing any system exception from the UOW that failed to be handled by its container.

When the outer business activity scope completes, its success or failure determines the completion direction (close or compensate) of any active compensation handlers that are owned by the outer business activity scope, including those promoted by the successful completion of inner business activity scopes. If the outer business activity scope completes successfully, it drives all active compensation handlers to close. If the outer business activity scope fails, it drives all active compensation handlers to compensate.

This compensation behavior is summarized in the following table.

Table 54. Compensation behavior for a single business activity scope

Inner business activity scope	Outer business activity scope	Compensation behavior
Succeeds	Succeeds	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW succeeds, the outer business activity scope drives all compensation handlers to close.
Fails	Succeeds	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is caught, when the outer UOW succeeds, the outer business activity scope drives all remaining active compensation handlers to close.
Fails	Fails	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is not caught, the outer business activity scope fails. When the outer business activity scope fails, either because of the unhandled exception or for some other reason, all remaining active compensation handlers are compensated.
Succeeds	Fails	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW fails, the outer business activity scope drives all compensation handlers to compensate.

When a UOW with an associated business activity scope completes, the business activity scope always completes in the same direction as the UOW that it is associated with. The only way that you can influence the direction of the business activity scope is to influence the UOW that it is associated with, which you can do by using the setCompensateOnly method of the business activity API.

A compensation handler that is registered within a transactional UOW might be inactive initially, depending on the method invoked from the business activity API. Inactive handlers in this situation become active when the UOW in which that handler is declared completes successfully. A compensation handler that is registered outside a transactional UOW always becomes active immediately. For more information, see “The business activity API” on page 1858.

Each business activity scope in the diagram represents a business activity. For example, the outer business activity running under BAScope1 can be a holiday booking scenario, with BAScope2 being a flight booking activity and BAScope3 a hotel booking. If either the flight or hotel bookings fail, the overall holiday booking by default also fails. Alternatively if, for example, the flight booking fails, you might want your application to try booking a flight using another component that represents a different airline. If the overall holiday booking fails, the application can use compensation handlers to cancel any flights or hotels that are already successfully booked.

Use of business activity scopes by application components

Application components do not use business activity scopes by default. You use the WebSphere Application Server assembly tools to specify the use of a business activity scope and to identify any compensation handler class for the component:

Default configuration

If a business activity context is present on a request received by a component with no business activity scope configuration, the context is stored by the container but never used during the method scope of the target component. A new business activity scope is not created. If the target component invokes another component, the stored business activity context is propagated and can be used by other compensating components.

Run enterprise bean methods under a business activity scope

Any business activity context present on the incoming request is received by the container and made available to the target component. If a new UOW is created for the target method, for example because the enterprise bean method has a **Transaction** setting of **Requires new**, the received business activity scope becomes an outer business activity scope to a newly created business activity. If the UOW is propagated from the calling component and used by the method, then the received business activity scope is used by the method. If a business activity scope does not exist on the invocation, a new business activity scope is created and used by the method.

To create a business activity scope when an enterprise bean is invoked, you must configure the enterprise bean to run enterprise bean methods under a business activity scope. You must also configure the deployment descriptors for the method being invoked, to specify the creation of a new UOW upon invocation. For instructions on how to perform these actions, see Creating an application that exploits the business activity support.

The effect of application server shutdown on active transactions and later recovery

When an application server shuts down, any active transactions are rolled back. If all transactions are successfully completed in this way, message WTRN0105I is logged, and on the next server restart no recovery activity is needed. If message CWWTR0105I is not logged for an application server shutdown, this does not indicate that there has been a failure, only that recovery activity is required when the server restarts.

A clean shutdown of all application servers should be achieved before the product is uninstalled, to avoid data integrity problems.

Extended JTA support

Extended JTA support provides application programming interfaces additional to the UserTransaction interface that is defined in the JTA as part of the J2EE specification. Specifically, the API extensions provide the following functionality:

- Access to global and local transaction identifiers associated with the thread.

  The global id is based on the tid in CosTransactions::PropagationContext: and the local id identifies the transaction uniquely within the local JVM.
- A transaction synchronization callback that enables any J2EE component to register an interest in transaction completion.

  This can be used by advanced applications to flush updates before transaction completion and clear up state after transaction completion. J2EE (and related) specifications position this function generally as the domain of the J2EE containers.

An application uses a JNDI lookup of java:comp/websphere/ExtendedJTATransaction to get an ExtendedJTATransaction object, which it then uses as follows:

```
ExtendedJTATransaction exJTA = (ExtendedJTATransaction)ctx.lookup("
 java:comp/websphere/ExtendedJTATransaction");
SynchronizationCallback sync = new SynchronizationCallback();
exJTA.registerSynchronizationCallback(sync);
```

The ExtendedJTATransaction object supports the registration of one or more application-provided SynchronizationCallbacks. Depending on how the callback is registered, each registered callback is called at one of the following points:

- At the end of every transaction that runs on the application server (whether the transaction is started locally or imported).
- At the end of the transaction for which the callback was registered.

The following information provides an overview of the interfaces provided by the Extended JTA support. For more detailed information, see the Javadoc.

SynchronizationCallback interface

An object implementing this interface is enlisted once through the ExtendedJTATransaction interface, and receives notification of transaction completion.

Although an object implementing this interface can run in a J2EE server, there is no specific J2EE component active when this object is called. So, the object has limited direct access to any J2EE resources. Specifically, it has no access to the java: namespace or to any container-mediated resource. Such an object can cache a reference to a J2EE component (for example, a stateless session bean) that it delegates to. The object would then have all the normal access to J2EE resources and could be used, for example, to acquire a JDBC connection and flush updates to a database during beforeCompletion.

ExtendedJTATransaction interface

A WebSphere programming model extension to the J2EE JTA support. An object implementing this interface is bound, by WebSphere J2EE containers that support this interface, at java:comp/websphere/ExtendedJTATransaction. Access to this object, when called from an EJB container, is not restricted to component-managed transactions.

Web Services Atomic Transaction support in WebSphere Application Server

The Web Services Atomic Transaction support in WebSphere Application Server provides transactional quality of service to the Web services environment. This enables distributed Web service applications, and the resources they use, to take part in distributed global transactions.

The Web Services Atomic Transaction (WS-AT) support is an implementation of the following specifications on WebSphere Application Server. These specifications define a set of Web services that enable Web service applications to participate in global transactions distributed across the heterogeneous Web service environment.

- Web Services Atomic Transaction (WS-AT), at http://www-106.ibm.com/developerworks/webservices/library/ws-atomtran/

  WS-AT is a specific coordination type that defines protocols for atomic transactions.
- Web Service Coordination (WS-COOR), at http://www-106.ibm.com/developerworks/webservices/library/ws-coor/

  WS-COOR specifies a CoordinationContext and a Registration service with which Participant web services may enlist to take part in the protocols offered by specific coordination types.

The WS-AT support is an interoperability protocol that introduces no new programming interfaces for transactional support. Global transaction demarcation is provided by standard J2EE use of the JTA UserTransaction interface. If a Web service request is made by an application component running under a global transaction, then a WS-AT CoordinationContext is implicitly propagated to the target Web service, but only if the appropriate application deployment descriptors have been set as described in Configuring transactional deployment attributes.

If WebSphere Application Server is the system hosting the target endpoint for a Web service request that contains a WS-AT CoordinationContext, then WebSphere automatically establishes a subordinate JTA transaction in the target runtime environment that becomes the transactional context under which the target Web service application executes.

The following figure, Figure 19 on page 1852, shows a transaction context shared between two WebSphere application servers for a Web service request that contains a WS-AT CoordinationContext.

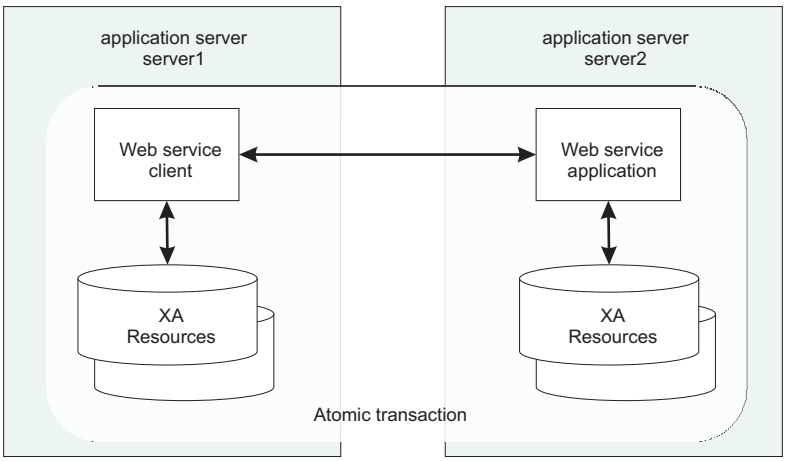


Figure 19. Transaction context shared between two WebSphere application servers.

WS-AT support restrictions

In this version of WebSphere Application Server, WS-AT contexts cannot be started from a non-recoverable client process.

Application design considerations

WS-AT is a two-phase commit transaction protocol and is suitable for short duration transactions only.

Because the purpose of an atomic transaction is to coordinate resource managers that isolate transactional updates by holding transactional locks on resources, it is generally not recommended that WS-AT transactions be distributed across enterprise domains. Inter-enterprise transactions typically require a looser semantic than two-phase commit and, in such scenarios, it can be more appropriate to use a compensating business transaction, for example a Web Services Business Activity or as part of a BPEL process.

WS-AT is most appropriate for distributing transaction context across Web services deployed within a single enterprise. Only request-response message exchange patterns carry transaction context since the originator (application or container) of a transaction needs to be sure that all business tasks executed under that transaction have finished before requesting the completion of a transaction. Web services invoked by a one-way request never run under the transaction of the requesting client.

Application development considerations

There are no specific development tasks required for Web service applications to take advantage of WS-AT. There are some application deployment descriptors that need to be set appropriately, as described in Configuring transactional deployment attributes.

Application developers do not need to explicitly register WS-AT participants. The WebSphere Application Server runtime takes responsibility for the registration of WS-AT participants, in the same way as the registration of XAResources in the JTA transaction to which the WS-AT transaction is federated. At transaction completion time, all XAResources and WS-AT participants are atomically coordinated by the WebSphere Application Server transaction service.

If a JTA transaction is active on the thread when a Web service Application request is made, the transaction is propagated across on the Web service request and established in the target environment. This is analogous to the distribution of transaction context over IIOP as described in the EJB specification. Any transactional work performed in the target environment becomes part of the same global transaction.

Support for Web Services protocols

WebSphere Application Server supports a number of Web Services protocols. These protocols provide standard ways of defining Web service applications, allowing the applications to operate independently of the product, platform or programming language used.

Web Services protocols are defined by the Oasis group. Refer to the Oasis Web site, http://www.oasis-open.org, for specifications and further information on each protocol. Use the sub-topics below to find out more information about the support for each protocol in WebSphere Application Server.

- "Web Services Atomic Transaction support in WebSphere Application Server" on page 1851
- "Web Services Business Activity support in WebSphere Application Server" on page 1855
- "Web Services transactions, firewalls and intermediary nodes" on page 1855

Web Services Atomic Transaction support in WebSphere Application Server:

The Web Services Atomic Transaction support in WebSphere Application Server provides transactional quality of service to the Web services environment. This enables distributed Web service applications, and the resources they use, to take part in distributed global transactions.

The Web Services Atomic Transaction (WS-AT) support is an implementation of the following specifications on WebSphere Application Server. These specifications define a set of Web services that enable Web service applications to participate in global transactions distributed across the heterogeneous Web service environment.

- Web Services Atomic Transaction (WS-AT), at http://www-106.ibm.com/developerworks/webservices/library/ws-atomtran/

  WS-AT is a specific coordination type that defines protocols for atomic transactions.
- Web Service Coordination (WS-COOR), at http://www-106.ibm.com/developerworks/webservices/library/ws-coor/

  WS-COOR specifies a CoordinationContext and a Registration service with which Participant web services may enlist to take part in the protocols offered by specific coordination types.

The WS-AT support is an interoperability protocol that introduces no new programming interfaces for transactional support. Global transaction demarcation is provided by standard J2EE use of the JTA UserTransaction interface. If a Web service request is made by an application component running under a global transaction, then a WS-AT CoordinationContext is implicitly propagated to the target Web service, but only if the appropriate application deployment descriptors have been set as described in Configuring transactional deployment attributes.

If WebSphere Application Server is the system hosting the target endpoint for a Web service request that contains a WS-AT CoordinationContext, then WebSphere automatically establishes a subordinate JTA transaction in the target runtime environment that becomes the transactional context under which the target Web service application executes.

The following figure, Figure 19 on page 1852, shows a transaction context shared between two WebSphere application servers for a Web service request that contains a WS-AT CoordinationContext.

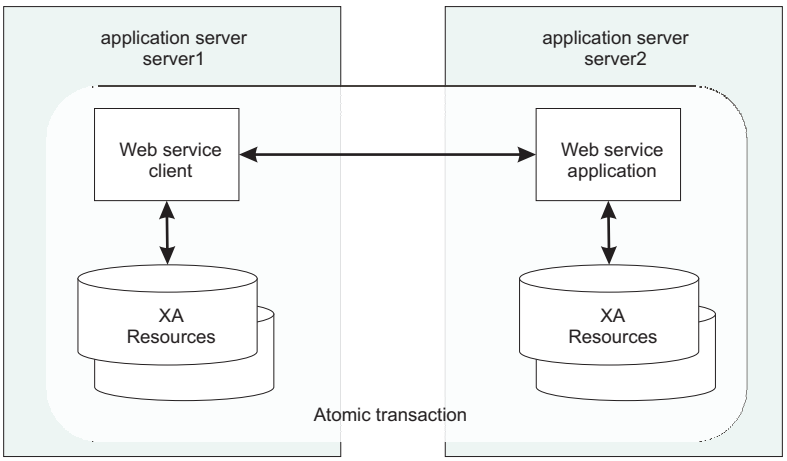


Figure 20. Transaction context shared between two WebSphere application servers.

WS-AT support restrictions

In this version of WebSphere Application Server, WS-AT contexts cannot be started from a non-recoverable client process.

Application design considerations

WS-AT is a two-phase commit transaction protocol and is suitable for short duration transactions only.

Because the purpose of an atomic transaction is to coordinate resource managers that isolate transactional updates by holding transactional locks on resources, it is generally not recommended that WS-AT transactions be distributed across enterprise domains. Inter-enterprise transactions typically require a looser semantic than two-phase commit and, in such scenarios, it can be more appropriate to use a compensating business transaction, for example a Web Services Business Activity or as part of a BPEL process.

WS-AT is most appropriate for distributing transaction context across Web services deployed within a single enterprise. Only request-response message exchange patterns carry transaction context since the originator (application or container) of a transaction needs to be sure that all business tasks executed under that transaction have finished before requesting the completion of a transaction. Web services invoked by a one-way request never run under the transaction of the requesting client.

Application development considerations

There are no specific development tasks required for Web service applications to take advantage of WS-AT. There are some application deployment descriptors that need to be set appropriately, as described in Configuring transactional deployment attributes.

Application developers do not need to explicitly register WS-AT participants. The WebSphere Application Server runtime takes responsibility for the registration of WS-AT participants, in the same way as the registration of XAResources in the JTA transaction to which the WS-AT transaction is federated. At transaction completion time, all XAResources and WS-AT participants are atomically coordinated by the WebSphere Application Server transaction service.

If a JTA transaction is active on the thread when a Web service Application request is made, the transaction is propagated across on the Web service request and established in the target environment. This is analogous to the distribution of transaction context over IIOP as described in the EJB specification. Any transactional work performed in the target environment becomes part of the same global transaction.

Web Services Business Activity support in WebSphere Application Server:

With Web Services Business Activity (WS-BA) support in WebSphere Application Server, Web services on disparate systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error.

The Web Services Business Activity (WS-BA) support is an implementation of the following specifications in WebSphere Application Server. These specifications define a set of protocols that enable Web service applications to participate in loosely coupled business processes that are distributed across the heterogeneous Web service environment, with the ability to compensate actions if an error occurs. For example, an application that sends an e-mail cannot unsend it following a failure. The application can, however, provide a business-level compensation handler that sends another e-mail advising of the new circumstances. A *business activity* is a group of general tasks that you want to link together so that the tasks have an agreed outcome.

- Web Services Business Activity (WS-BA), at http://www-106.ibm.com/developerworks/library/specification/ws-tx/#ba

  WS-BA is a specific coordination type that defines protocols for business activities.
- Web Service Coordination (WS-COOR), at http://www-106.ibm.com/developerworks/webservices/library/ws-coor/

  WS-COOR specifies a CoordinationContext and a registration service with which participant Web services can enlist to take part in the protocols that are offered by specific coordination types.

In addition to supporting the WS-BA interoperability protocol, WebSphere Application Server provides a programming interface for creating business activities and compensation handlers. With this programming interface, you can specify compensation data and check or alter the status of a business activity.

You can also use this compensation ability with applications that are not Web services, as long as these applications involve communication between WebSphere Application Servers only. See the related topics for more information.

Web Services transactions, firewalls and intermediary nodes:

You can configure your system to enable propagation of Web Services Atomic Transactions (WS-AT) message contexts and Web Service Business Activities (WS-BA) message contexts across firewalls or outside the WebSphere Application Server domain. With this configuration you can distribute Web service applications that use WS-AT or WS-BA across disparate systems. The topology that you use can have an effect on the high availability and affinity behavior of the transactions.

The topologies that are available to you are as follows:

Direct connection

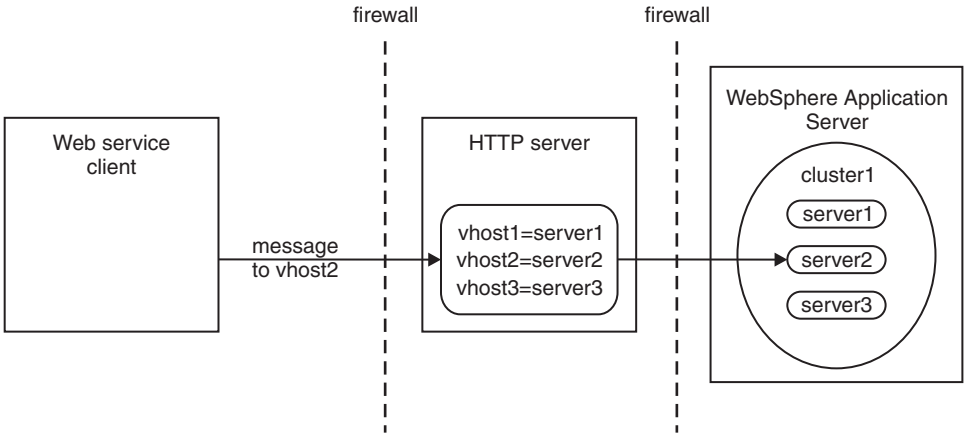
No intermediary node exists in this topology. The client communicates directly with a specific WebSphere Application Server. This topology supports high availability for transactions on requests made within a cluster, and affinity of transactions.

HTTP server, such as IBM HTTP Server

In this topology, the client communicates with an HTTP server, which always routes the client requests to a specific server. You configure the HTTP server to specify the WebSphere Application Server to route requests to.

The HTTP server cannot provide either affinity or high availability for transactions. However, transactional integrity is assured because recovery processing occurs after the failed server restarts.

Note: You can still enable high availability on the WebSphere Application Server. Clients accessing this server through an HTTP server cannot have high availability of transactions, however other clients accessing the same server can.



Use of local transactions

Local transaction containment (LTC) support, and its configuration through local transaction extended deployment descriptors, gives IBM WebSphere Application Server application programmers a number of advantages. This topic describes those advantages and how they relate to the settings of the local transaction extended deployment descriptors. This topic also describes points to consider to help you best configure transaction support for some example scenarios that use local transactions.

Develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not need to use global transactions, it is often more efficient to deploy the bean with the Container Transaction deployment descriptor **Transaction** attribute set to `Not supported` instead of `Required`.

With the extended local transaction support of IBM WebSphere Application Server, applications can perform the same business logic in an unspecific transaction context as they can under a global transaction. An enterprise bean, for example, runs under an unspecified transaction context if it is deployed with a **Transaction** attribute of `Not supported` or `Never`.

The extended local transaction support provides a container-managed, implicit local transaction boundary within which application updates can be committed and their connections cleaned up by the container. Applications can then be designed with a greater degree of independence from deployment concerns. This makes using a **Transaction** attribute of `Supports` much simpler, for example, when the business logic may be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage regardless of whether or not the application runs under a transaction. The application can depend on the close behaving in the same way and not causing a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios running business logic under a **Transaction** policy of `Not supported` performs better than if it had been run under a `Required` policy. This benefit is exploited through the **Local Transactions - Resolution-control** extended deployment setting of `ContainerAtBoundary`. With this setting, application interactions with resource providers (such as databases) are managed within implicit RMLTs that are both started and ended by the container. The RMLTs are committed by the container at the configured **Local Transactions - Boundary**; for example at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

Use local transactions in a managed environment that guarantees clean-up.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default **Local Transactions - Resolution-control** extended deployment setting of `Application`. In this case, the container ensures connection cleanup at the boundary of the local transaction context.

J2EE specifications that describe application use of local transactions do so in the manner provided by the default setting of **Local Transactions - Resolution-control**=`Application` and **Local Transactions - Unresolved-action**=`Rollback`. By configuring the **Local Transactions - Unresolved-action** extended deployment setting to `Commit`, then any RMLTs started by the application but not completed when the local transaction containment ends (for example, when the method ends) are committed by the container. This usage applies to both servlets and enterprise beans.

Extend the duration of a local transaction beyond the duration of an EJB component method.

The J2EE specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. You can exploit the **Local Transactions - Boundary** extended deployment setting to give the following advantages:

- Significantly extend the use-cases of RMLTs
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

You can use an ActivitySession to provide a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which can be controlled by a client. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is exploited through the **Local Transactions - Boundary** extended deployment setting of `ActivitySession`. Such extended RMLTs can remain under the control of the application or be managed by the container depending on the use of the **Local Transactions - Resolution-control** deployment descriptor setting.

Coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can exploit ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans under that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

To determine how best to configure the transaction support for an application, depending on what you want to do with transactions, consider the following points.

General points

- You want to start and end global transactions explicitly in the application (BMT session beans and servlets only).

  For a session bean, set the **Transaction type** to `Bean` (to use bean-managed transactions) in the component's deployment descriptor. (You do not need to do this for servlets.)
- You want to access only one XA or non-XA resource in a method.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Supports`.
- You want to access several XA resources atomically across one or more bean methods.

  In the Container transaction deployment descriptor, set **Transaction** to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resource in a method without having to worry about managing your own local transactions.

In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.

- You want to access several non-XA resources in a method and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application` and set **Local Transactions - Unresolved-action** to **Rollback**. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.
- You want to access one of more non-XA resources across multiple EJB method calls without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container Transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to use a single non-XA resource and one or more XAResources.

  Use the *Last Participant Support*.

The business activity API

Use the business activity API to create business activities and compensation handlers for an application component, and to log data that is required for compensating an activity in the event of a failure.

Overview

The business activity support provides a UserBusinessActivity API, a CompensationHandler interface and two exceptions: RetryCompensationHandlerException and CompensationHandlerFailedException. You can look up the UserBusinessActivity interface from the application server Java Naming and Directory Interface (JNDI) at `java:comp/websphere/UserBusinessActivity`. For example:

```
InitialContext ctx = new InitialContext();
UserBusinessActivity uba = (UserBusinessActivity) ctx.lookup("java:comp/websphere/UserBusinessActivity");
```

If an application component is running work that might require compensating upon failure, you must provide a compensation handler class that is assembled as part of the deployed application. This new Java class must implement the com.ibm.websphere.wsba.CompensationHandler interface, which supports the close and compensate methods, which take a parameter of a Service Data Object (SDO). During normal application processing, the application can make one or more invocations to the `setCompensationDataImmediate` or `setCompensationDataAtCommit` methods, passing in an SDO representing the current state of the work performed.

When the underlying unit of work (UOW) that the root business activity is associated with completes, all registered compensators are coordinated to complete. During completion, either the compensate or the close method is called on the compensation handler, passing in the most recent compensation data logged by the application component as a parameter. Your compensation handler implementation must be able to understand the data that is stored in the SDO DataObject; using this data, the compensation handler must be able to determine the nature of the work performed by the enterprise bean and compensate or close in an appropriate manner, for example by undoing changes made to database rows in the event of a failure. You associate the compensation handler with an application component by using the assembly tooling, such as Rational Application Developer.

Active and inactive compensation handlers

You implement the CompensationHandler interface for any application component that executes code that might need to be compensated within a business activity scope. CompensationHandler objects are registered implicitly with the business activity scope under which the application runs, whenever the application calls the UserBusinessActivity API to specify compensation data. Compensation handlers can be in one of two states, active or inactive, depending on any transactional UOW under which they are registered. A compensation handler registered within a transactional UOW might be initially inactive until the transaction commits, at which point it becomes active (see below). A compensation handler registered outside a transactional UOW always becomes active immediately.

When a business activity completes, it drives active compensation handlers only. Any inactive compensation handlers that are associated with the business activity are discarded and never driven.

Logging compensation data

The business activity API specifies two methods that allow the application to log compensation data. This data is made available to the compensation handlers during their processing upon completion of the business activity. The application calls one of these methods, depending on whether it expects transactions to be part of the business activity.

setCompensationDataAtCommit()

Call this method if the application expects a global transaction on the thread.

- If a global transaction is present on the thread, the CompensationHandler object is initially inactive. If the global transaction fails, it rolls back any transactional work done within its transaction context in an atomic manner, and drives the business activity to compensate other completed UOWs. The compensation handler does not need to be involved. If the global transaction commits successfully, the compensation handler becomes active because it is required to compensate the durable work that is completed by the global transaction, if the overall business activity fails. The setCompensationDataAtCommit method configures the CompensationHandler instance to perform this compensation function.
- If a global transaction is not present when this method is called, the compensation handler becomes active immediately.

For example, using the same business activity instance as in the previous example:

```
DataObject compensationData = doWorkWhichWouldNeedCompensating();
uba.setCompensationDataAtCommit(compensationData);
```

setCompensationDataImmediate()

Call this method if the application does not expect a global transaction on the thread.

The setCompensationDataImmediate method makes a CompensationHandler instance active immediately, regardless of the current UOW context at the time that the method is invoked. The compensation handler is always able to participate during completion of the business activity.

The role of the setCompensationDataImmediate method is to compensate any non-transactional work, in other words, work that can be performed either inside or outside a global transaction, but is not governed by the transaction. For example, sending an e-mail. The compensation handler must be active immediately so if a failure occurs within a business activity, this non-transactional work is always compensated.

Although these two compensation data logging methods, if called within the same enterprise bean, use the same CompensationHandler class, they create two separate instances of the CompensationHandler class at run time. Therefore, the actions of the methods are mutually exclusive; calling one of the methods does not overwrite any work carried out by the other method.

A CompensationHandler instance is not added to a business activity automatically when a business-activity-enabled enterprise bean is invoked. The association is done explicitly through the business activity API, using one of the compensation data logging methods described previously. If such a method is called passing in `null` as a parameter, the CompensationHandler instance is added to the current business activity, and `null` is passed later to either the `compensate` or the close method upon completion of the business activity scope. With this process, you can add the compensation handler to the business activity, and code default behavior within the compensation handler. It is your responsibility to handle the case of `null`s that pass back to your CompensationHandler instance. This scenario happens only if you call a compensation data logging method with `null` as a parameter.

As described previously, the business activity support adds a CompensationHandler instance to the business activity when a compensation data logging method is called for the first time by the enterprise bean using that business activity. At the same time, a snapshot of the J2EE context is taken and logged with the compensation data. Upon completion of the business activity, all the compensation handlers that were added to the business activity are driven to compensate or close. The code that you create within the CompensationHandler class is guaranteed to run wiithin the same J2EE context that was captured in the earlier snapshot.

For details about the methods available in the business activity API, see the WebSphere Application Server application programming interface reference information.

Transaction service exceptions

This topic lists the exceptions that can be thrown by the WebSphere Application Server transaction service. The exceptions are listed in the following groups:
- Standard exceptions
- Heuristic exceptions

If the EJB container catches a system exception from the business method of an enterprise bean, and the method is running within a container-managed transaction, the container rolls back the transaction before passing the exception on to the client. For more information about how the container handles the exceptions thrown by the business methods for beans with container-managed transaction demarcation, see the section *Exception handling* in the Enterprise JavaBeans 2.0 specification. That section specifies the container's action as a function of the condition under which the business method executes and the exception thrown by the business method. It also illustrates the exception that the client receives and how the client can recover from the exception.

Standard exceptions

The standard exceptions such as TransactionRequiredException, TransactionRolledbackException, and InvalidTransactionException are defined in the Java Transaction API (JTA) 1.0.1 Specification.

InvalidTransactionException
: This exception indicates that the request carried an invalid transaction context.

TransactionRequiredException exception
: This exception indicates that a request carried a null transaction context, but the target object requires an active transaction.

TransactionRolledbackException exception
: This exception indicates that the transaction associated with processing of the request has been rolled back, or marked for roll back. Thus the requested operation either could not be performed or was not performed because further computation on behalf of the transaction would be fruitless.

Heuristic exceptions

A heuristic decision is a unilateral decision made by one or more participants in a transaction to commit or rollback updates without first obtaining the consensus outcome determined by the Transaction Service. Heuristic decisions are an issue only after the participant has been prepared and the second phase of

commit processing is underway. Heuristic decisions are normally made only in unusual circumstances, such as repeated failures by the transaction manager to communicate with a resource manage during two-phase commit. If a heuristic decision is taken, there is a risk that the decision differs from the consensus outcome, resulting in a loss of data integrity.

The following list provides a summary of the heuristic exceptions. For more detail, see the Java Transaction API (JTA) 1.0.1 Specification.

HeuristicRollback exception
: This exception is raised on the commit operation to report that a heuristic decision was made and that all relevant updates have been rolled back.

HeuristicMixed exception
: This exception is raised on the commit operation to report that a heuristic decision was made and that some relevant updates have been committed and others have been rolled back.

UserTransaction interface - methods available

For details about the methods available with the UserTransaction interface, see the WebSphere Application Server application programming interface reference information (Javadoc) or the Java Transaction API (JTA) 1.0.1 Specification.

Configuring transaction properties for an application server

Use this task to change the transaction log properties, to move your transaction logs to a new location, or to update the parameters for the server's transaction logs.

You might want to move to your logs to a different storage device. Perform this task when you are ready to move your transaction logs or when you need to make a change to the parameters. You must restart the application server to make configuration changes take effect.

To configure the transaction properties for an application server, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, select **Servers-> Manage Application Servers-> *your_app_server*** This displays the properties of the application server, *your_app_server*, in the content pane.
3. Under Container Settings, expand Container Services, then click Transaction Service to display the properties page for the transaction service, as two notebook pages:
   Configuration
   : The values of properties defined in the configuration file. If you change these properties, the new values are applied when the application server next starts.

   Runtime
   : The runtime values of properties. If you change these properties, the new values are applied immediately, but are overwritten with the Configuration values when the application server next starts.
4. To review transaction-related configuration properties, ensure that the Configuration page is displayed.
5. **Optional:** If you want to change the directory in which transaction logs are written, type the full pathname of the directory in the **Transaction log directory** field. You can check the current runtime value of **Transaction log directory**, by clicking the Runtime tab.

   When using WebSphere Application Server without High Availability support, you can leave the recovery log configuration for persistent services (such as the transactions service) unset. The application server assumes a default location within the appropriate profile directory. When High Availability support is enabled, this default may not be visible from all servers in the cluster (for example, if they are in different profiles or physical nodes.) As a result, it is recommended that the recovery log location be configured for each server in the cluster before enabling High Availability.

   You can also specify a size for the transaction logs, as described in the following step.

Note: If you change the transaction log directory, you should apply the change and restart the application server as soon as possible, to minimize the risk of problems occurring before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server next starts with the new log directory and is unable to automatically resolve in-flight transactions that were recorded in the old log directory.

6. **Optional:** If you want to change the default file size of transaction log files, modify the **Transaction log directory** field to include a file size setting, in the following format:

   directory_name;*file_size*

   Where
   - *directory_name* is the name of the transaction log directory
   - *file_size* is the new default size specified in bytes. The *n*K or *n*M suffix can be used to indicate kilobytes or megabytes. If you do not specify a file size value, the default value of 1M is used.

   For example (Windows systems), `c:\tranlogs;2M` indicates the files are to be created with 2M bytes size and stored in the directory c:\tranlogs.

   In a non-production environment, you can use the transaction log directory value of `;0` to disable transaction logging. (There must be no directory name element before the size element of 0.) You should not disable transaction logging in a production environment because this prevents recovery after a system failure, and therefore data integrity cannot be guaranteed.

7. **Optional:** Review or change the value of transaction timeout properties:

   Total transaction lifetime timeout
   Type the number of seconds a transaction can remain inactive before it is ended by the transaction service. A value of 0 (zero) indicates that there is no timeout limit.

   Client inactivity timeout
   Type the number of seconds after which a client is considered inactive and the transaction service ends any transactions associated with that client. A value of 0 (zero) indicates that there is no timeout limit.

8. **Optional:** Review or change heuristic-related properties:

   Heuristic retry limit
   The number of times that the application server retries a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

   Heuristic retry wait
   The number of seconds that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

   Enable logging for heuristic reporting
   Select this property to enable the application server to log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

   Heuristic completion direction
   Select the direction used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or depends on manual completion by the administrator.

9. Review or change other configuration properties, to suit your requirements. For more information about the properties of the transaction service, see “Transaction service settings” on page 1863.
10. Click **OK** and save.
11. Stop then restart the application server.

    If you change the transaction log directory configuration property to an incorrect directory name, the application server will restart but be unable to open the transaction logs. You should change the configuration property to a valid directory name, then restart the application server.

If you are running the application server as non-root, modify the permissions on the new transaction log location. If you want to use peer recovery of transactions on a shared device with non-root users, make sure that your non-root users and groups have matching identification numbers across machines

Transaction service settings

Use this page to specify settings for the transaction service. The transaction service is a server runtime component that can coordinate updates to multiple resource managers to ensure atomic updates of data. Transactions are started and ended by applications or the container in which the applications are deployed.

To view this administrative console page, click **Servers > Application Servers >** *server_name* > **Container Services > Transaction Service**.

Related concepts

"Web Services transactions, firewalls and intermediary nodes" on page 1855
You can configure your system to enable propagation of Web Services Atomic Transactions (WS-AT) message contexts and Web Service Business Activities (WS-BA) message contexts across firewalls or outside the WebSphere Application Server domain. With this configuration you can distribute Web service applications that use WS-AT or WS-BA across disparate systems. The topology that you use can have an effect on the high availability and affinity behavior of the transactions.

Transaction log directory

Specifies the name of a directory for this server where the transaction service stores log files for recovery.

If you do not specify this directory during server configuration, the transaction log uses a default that is based on your installation directory: *app_server_root*)/ tranlog/*cell_ name*/*node_ name*/*server_ name*.

When an application running on WebSphere Application Server accesses more than one resource, Application Server stores transaction information within the product directory to properly coordinate and manage the distributed transaction. In a higher transaction load, this persistence slows down performance of the application server due to its dependency on the operating system and the underlying storage systems. To achieve better performance, designate a new directory for the log files on a separate, physically larger storage system.

Set this property to change the log file directory for an application server only if the applications use distributed resources or XA transactions; for example, multiple databases and resources are accessed within a single transaction.

If your application server demonstrates one or more of the following symptoms, change log file directories.
- CPU utilization remains low despite an increase in transactions
- Transactions fail with several timeouts
- Transaction rollbacks occur with *unable to enlist transaction* exception
- The application server hangs in the middle of a run and requires the server to be restarted
- The disk on which an application server is running shows higher utilization

File system recommendations:

- Store log files on a redundant array of independent disks (RAID)

  In RAID configurations, the task of writing data to the physical media is shared across the multiple drives. This technique yields more concurrent access to storage for persisting transaction information, and faster access to that data from the logs. Depending upon the design of the application and storage subsystem, performance gains can range from 10% to 100%, or even more in some cases.
- Do not store log files with the operation system I/O mode set to concurrent I/O (CIO)

  When you designate a transaction log directory, ensure that the file system uses only synchronous write-through and write serialization operations. Some operating systems, such as AIX JFS2, support an

optional concurrent I/O (CIO) mode whereby the file system does not enforce serialization of write operations. On these systems, do not use CIO mode for Application Server transaction recovery log files.

Data type	String
Default	Initial value is the *app_server_root*/tranlog/*cell_name*/*node_name*/*server_name* directory and a default size of 1MB.
Recommended	Create a file system with at least 3-4 disk drives raided together in a RAID-0 configuration. Then, create the transaction log on this file system with the default size. When the server is running under load, check the disk input and output. If disk input and output time is more then 5%, consider adding more physical disks to lower the value.

If you migrate a WebSphere Application Server Version 5 node to Version 6, the stored location of this configuration property is moved from the server level to the node (server index) level. If you have specified a non-default log directory for a Version 5 application server, you are prompted to save the transaction service settings again, to confirm that you want the log directory saved to the node level.

Total transaction lifetime timeout

Specifies the default maximum time, in seconds, allowed for transactions started on this server to complete. Any such transactions that do not complete before this timeout occurs are rolled back.

If you set this value to 0, only the maximum transaction timeout configuration value applies.

Data type	Integer
Units	Seconds
Default	120
Range	0 to 2 147 483 647

Asynchronous response timeout

Specifies the amount of time, in seconds, that the server waits for responses to WS-AT protocol messages.

Data type	Integer
Units	Seconds
Default	30
Range	0 to 2 147 483 647

Client inactivity timeout

Specifies the maximum duration, in seconds, between transactional requests from a remote client. Any period of client inactivity that exceeds this timeout results in the transaction being rolled back in this application server.

If you set this value to 0, there is no timeout limit.

Data type	Integer
Units	Seconds
Default	60
Range	0 to 2 147 483 647

Maximum transaction timeout

Specifies the maximum time to complete, in seconds, for transactions that run in this server. This value should be greater than or equal to the total transaction timeout.

This limit constrains transaction timeout set programmatically by BMT beans and values imported with contexts received from foreign servers.

This value limits the upper bound of all other transaction related timeouts. For example, if a component attempts to set a transaction timeout of 360 seconds, and the Maximum Transaction Timeout setting is 300 seconds, the Maximum Transaction Timeout setting of 300 seconds is used.

If set to 0, there is no limit and therefore the timeout specified by the Total transaction lifetime timeout property or component timeout is used.

Data type	Integer
Units	Seconds
Default	300
Range	0 to 2 147 483 647

Heuristic retry limit

Specifies the number of times that the application server retries a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

If the application server abandons the retries, then the resource manager or remote partner is responsible for ensuring that the resource or partner's branch of the transaction is completed appropriately. The application server raises (on behalf of the resource or partner) an exception that indicates a heuristic hazard. If a commit was requested, the transaction originator receives an exception on the commit operation; if the transaction is container-initiated, then the container returns a remote exception or EJB exception to the EJB client.

Data type	Integer
Default	0
Range	0 to 2 147 483 647 A value of 0 (the default) means retry forever.

Heuristic retry wait

Specifies the number of seconds that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

Data type	Integer
Default	0
Range	0 to 2 147 483 647 A value of 0 means that the application server determines the retry wait; the server doubles the retry wait after every 10 failed retries.

Enable logging for heuristic reporting

Specifies whether the application server logs about-to-commit-one-phase-resource events from transactions that involve both a one-phase commit resource and two-phase commit resources.

This property enables logging for heuristic reporting. If applications are configured to allow one-phase commit resources to participate in two-phase commit transactions, reporting of heuristic outcomes that occur at application server failure requires extra information to be written to the transaction log. If enabled,

one additional log write is performed for any transaction that involves both one-phase and two-phase commit resources. No additional records are written for transactions that do not involve a one-phase commit resource.

Data type Check box
Default Cleared
Range

Cleared
The application server does not log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

Selected
The application server does log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

Heuristic completion direction

Specifies the direction that is used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or depends on manual completion by the administrator.

Data type Drop-down list
Default ROLLBACK
Range

COMMIT
The Application server heuristically commits the transaction.

ROLLBACK
The Application server heuristically rolls back the transaction.

MANUAL
The application server depends on an administrator to manually complete or roll back transactions with heuristic outcomes.

Enable file locking

Specifies whether the use of file locks is enabled when opening the transaction service recovery log.

If you enable this setting, a file lock will be obtained before accessing the transaction service recovery log files. File locking is used to ensure that, in a highly available WebSphere Application Server deployment, only one application server can access a particular transaction service recovery log at any one time. This setting has no effect in a standard deployment where you have not configured high availability support.

Attention: This setting requires a compatible network file system, such as network file system version 4, to operate correctly.

Data type Check box
Default Selected

Enable protocol security

Specifies whether the secure exchange of transaction service protocol messages is enabled.

This setting has no effect unless you enable WebSphere Application Server security on the server.

Data type	Check box
Default	Selected

HTTP proxy prefix

Specifies the prefix used by WS-AtomicTransaction and WS-BusinessActivity requests that are sent through an intermediary node using the HTTP protocol.

Set this field if you are using an intermediary node, such as an HTTP server or Proxy Server for WebSphere, to send requests that comply with the Web Services Atomic Transaction or Web Services Business Activity protocols. The format for the prefix is as follows:

`http://host_name:port`

where

- *host_name* is the host name used by clients to contact the Web service
- *port* is the port for which the service is configured to accept client requests

If the intermediary node is not a Proxy Server, the prefix must be unique for each server. If you are using a Proxy Server, prefixes can be the same for each server in a cluster, because the Proxy Server determines dynamically which server to forward the request to.

The HTTPS prefix will be used if WebSphere Application Server security is enabled and protocol security is enabled for the transaction service, otherwise the HTTP prefix will be used.

Data type	String
Default	None

HTTPS proxy prefix

Specifies the prefix used by WS-AtomicTransaction and WS-BusinessActivity requests that are sent through an intermediary node using the HTTPS protocol.

Set this field if you are using an intermediary node, such as an HTTP server or Proxy Server for WebSphere, to send requests that comply with the Web Services Atomic Transaction or Web Services Business Activity protocols. The format for the prefix is as follows:

`https://host_name:port`

where

- *host_name* is the host name used by clients to contact the Web service
- *port* is the port for which the service is configured to accept client requests

If the intermediary node is not a Proxy Server, the prefix must be unique for each server. If you are using a Proxy Server, prefixes can be the same for each server in a cluster, because the Proxy Server determines dynamically which server to forward the request to.

The HTTPS prefix will be used if WebSphere Application Server security is enabled and protocol security is enabled for the transaction service, otherwise the HTTP prefix will be used.

Data type	String
Default	None

Manual transactions

Specifies the number of transactions that await manual completion by an administrator.

If there are transactions awaiting manual completion, you can click the **Review** link to display a list of those transactions on the Transactions needing manual completion panel.

Data type	Integer
Default	0

Retry transactions

Specifies the number of transactions with some resources being retried.

If there are transactions with resources being retried, you can click the **Review** link to display a list of those transactions on the Transactions retrying resources panel.

Data type	Integer
Default	0

Heuristic transactions

Specifies the number of transactions that have completed heuristically.

If there are transactions that have completed heuristically, you can click the **Review** link to display a list of those transactions on the Transactions with heuristic outcome panel.

Data type	Integer
Default	0

Imported prepared transactions

Specifies the number of transactions that are imported and prepared but not yet committed.

If there are transactions that have been imported and prepared but not yet committed, you can click the **Review** link to display a list of those transactions on the Transactions imported and prepared panel.

Data type	Integer
Default	0

Transactions needing manual completion

Use this page to review transactions that need manual completion.

It is unusual for transactions to require manual completion. A transaction needs manual completion in the following circumstances:

1. An application was exploiting the last participant support to coordinate a single one-phase capable resource and one or more two-phase capable resources.
2. A failure occurred during the commit of the one-phase capable resource.
3. The transaction service heuristic completion direction is set to **Manual**, in the transaction service settings.

An administrator reviewing transactions in this state can review the actual outcome of any one-phase resources, using facilities provided by the specific resource manager, then use this page to complete the transaction accordingly.

To view this administrative console page, click **Servers** → **Application Servers** → [Content pane] *server_name* → **[Container Settings] Container Services** → **Transaction Service** → **Runtime** → **Manual transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Global ID
The global identifier of the transaction.

Buttons

Commit
Heuristically commit the selected transactions.

Rollback
Heuristically roll back the selected transactions.

Transactions retrying resources

Use this page to review transactions with resources being retried.

If the transaction manager has prepared resources, but has lost contact with the resource managers before committing them or aborting them, then the transaction manager retries the commit or rollback requests to the affected resource managers. The number of times and frequency that the transaction manager retries such commit or rollback requests is configured on the **Heuristic retry limit** and **Heuristic retry wait** properties of the transaction service settings.

An administrator can use this page to make the transaction service abandon the retries of one or more transactions. If a resource manager cannot be contacted, WebSphere Application Server relegates that resource manager to an in-doubt (prepared) state. The administrator then needs to use mechanisms specific to the resource manager to resolve the in-doubt status.

To view this administrative console page, click **Servers** → **Application Servers >** [Content pane] *server_name* **> [Container Settings] Container Services** → **Transaction Service** → **Runtime** → **Retry transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Status
The status of the transaction, shown as an integer value. The values correspond to the following status:

0 - active
1 - marked for rollback
2 - prepared
3 - committed
4 - rolled back
5 - unknown
6 - none
7 - preparing
8 - committing
9 - rolling back

Global ID
The global identifier of the transaction.

Buttons

Finish Abandon retrying resources for the selected transactions.

Transactions with heuristic outcome

Use this page to review transactions that completed with a heuristic outcome.

The page is provided for information purposes only. After you have reviewed the information in this page, then the only action required is to remove the transactions from the list. If you do not remove a transaction from the list, it is kept in the list for three days or until the server is shut down, whichever occurs first.

To view this administrative console page, click **Servers** → **Application Servers** [Content pane] *server_name* [Container Settings] **Container Services** → **Transaction Service** → **Runtime** → **Heuristic transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Heuristic outcome
The outcome of the transaction.

Global ID
The global identifier of the transaction.

Buttons

Clear Remove the selected transactions from the list.

Transactions imported and prepared

Use this page to review transactions that have been imported and prepared but not yet committed.

Under normal circumstances no administrative action is required for any of the transactions listed on this page. This page lists those transactions that are in a prepared state, but are being directed by an external transaction manager (for example, another WebSphere application server) from a transaction context that has been propagated.

Under aberrant circumstances, however, an administrator can configure WebSphere Application Server to resolve the transactions listed on this page independent of the external transaction manager. (This step might be necessary if, for example, the external transaction manager has become unavailable for an unacceptable period of time.)

Note: If the completion direction (commit or rollback) chosen administratively differs from the eventual direction of the external transaction manager, then the overall outcome of the transaction is not atomic and data corruption can result.

To view this administrative console page, click **Servers** → **Application Servers** [Content pane] *server_name* → **[Container Settings] Container Services** → **Transaction Service** → **Runtime** → **Imported prepared transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Global ID
The global identifier of the transaction.

Buttons

Commit
Heuristically commit the selected transactions.

Rollback
Heuristically roll back the selected transactions.

Transaction resources
Use this page to review resources used by a transaction.

To view this administrative console page, click **Servers** → **Application Servers** [Content pane] *server_name* → **[Container Settings] Container Services** → **Transaction Service** → **Runtime** → *transaction_type local_ID*.

Where:

- *transaction_type* is one of:
  - **Manual transactions - Review**
  - **Retry transactions - Review**
  - **Heuristic transactions - Review**
  - **Imported prepared transactions - Review**
- *local_ID* is the local ID of the transaction (as an active link in the list of transactions).

The details displayed depend on the resource provider.

Managing active and prepared transactions

Use this task to manage active and prepared transactions that might need administrator action.

Under normal circumstances, transactions should run and complete (commit or rollback) automatically, without the need for intervention. However, in some circumstances, you may need to resolve a transaction manually. For example, you may want to rollback a transaction that has become stuck polling a resource manager that you know will not become available again within the desired timeframe.

Note: If you choose to complete a transaction on an application server, it is recorded as having completed in the transaction service logs for that server, so will not be eligible for recovery during server start up. If you complete a transaction, you are responsible for cleaning up any in-doubt transactions on the resource managers affected.

You can use the administrative console to display a snapshot of all the transactions in an application server that are in the following states:

Manual transactions
Transactions awaiting administrative completion. For each transaction, the local id or global id is displayed. You can choose to display information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to commit or rollback transactions in this state.

Retry transactions
Transactions with some resources being retried. For each transaction, the local id or global id is displayed, and whether the transaction is committing or rolling back. You can choose to display

information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to finish (abandon retrying) transactions in this state.

Heuristic transactions
Transactions that have completed heuristically. For each transaction, the local id or global id and the heuristic outcome is displayed. You can choose to display information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to clear the transaction from the list.

Imported prepared transactions
Transactions that have been imported and prepared but not yet committed. For each transaction, the local id or global id is displayed. You can choose to display information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to commit or rollback transactions in this state.

To manage the active and prepared transactions for an application server, use the administrative console to complete the following steps:

1. Display the Transaction Service runtime page for application server:
   a. In the navigation pane, click **Servers-> Application Servers**
   b. In the content pane, click the name of the application server
   c. In the content pane, click the **Runtime** tab.
   d. Under Additional Properties, click **Transaction Service**

   This displays values for the runtime properties of the transaction service, including the number of transactions in the active and prepared states.
2. To display a snapshot of the transactions in a specific state, click **Review** in the field label.
3. **Optional:** If you want to display information about the resources associated with a transaction, click the name of the transaction.
4. **Optional:** If you want to act on a transaction, select the check box provided on the entry for the transaction, then click one of the buttons provided. Alternatively, to act on all transactions, select the check box in the header of the transactions table, then click a button.

Managing transaction logging for optimum server availability

This topic describes some considerations and actions that you can use to manage transaction logging to help ensure that the availability of your application servers is optimized.

The transaction service writes information to the transaction log for every global transaction which involves two or more resources or is distributed across multiple servers. The transaction log is stored on disk and is used by the transaction service for recovery after a system or server crash. The transaction log for each application server consists of multiple subdirectories and files held in a single directory. You can change the directory that an application server uses to store the transaction log, as described in Configuring transaction properties for an application server.

When a global transaction is completed, the information in the transaction log is not needed anymore so is marked for deletion. Periodically, this redundant information is garbage collected and the space reused by new transactions. The log files are created of fixed size at server startup, thus no further disk space allocation is required during the lifetime of the server. The default allocation is suitable for around 4000 concurrent transactions.

If all the log space is in use when a transaction needs to save information, the transaction is rolled back and the message `CWWTR0083W: The transaction log is full. Transaction rolled back.` is reported to the system error log. No more transactions can commit until more log space is made available when existing active transactions complete.

You can monitor the number of concurrent global transactions by using the performance monitoring counters for transactions. The “Global transaction commit time” counter is a measure of how long a transaction takes to complete and, therefore, how long the log is in use by a transaction. If this value is high, then transactions are taking a long time to complete, which can be due to resource manager or network failures. If you ensure that this value is low, the log is more efficiently used and unlikely to become full.

You can change the default size of log files by updating the transaction log settings as described in Configuring transaction properties for an application server.

Configuring transaction aspects of servers for optimum availability

This topic describes some considerations and actions that you can take to configure transaction-related aspects of application servers, to optimize the availability of those servers.

This helps your transactions to complete or recover more quickly. After changing transaction-related properties of an application server, you must restart the server.

To configure transaction-related aspects of application servers for optimum availability, complete the following steps:

1. Store the transaction log files on a fast disk in a highly-available file system, such as a RAID device. The transaction log may need to be accessed by every global transaction and be used for transaction recovery after a crash. Therefore, the disk the log files are being written to should be on a highly-available file system, such as a RAID device.

   The performance of the disk also directly affects the transaction performance. In general, a global transaction makes two disk writes, one after the prepare phase when the outcome of the transaction is known (this information is forced to disk) and a further disk write at transaction completion. Therefore, the transaction logs should be placed on the fastest disks available and not make use of network mounted devices.
2. Mirror the transaction log files by using hardware disk mirroring or dual-ported disks. If log files have been mirrored or can be recovered, they can be used when restarting a failed server or moved to an another machine and another server started there to perform recovery.

   Hardware disk mirroring or dual-ported disks can be used by specifying the appropriate file system directory for the transaction logs using the WebSphere Administrative Console.
3. Specify the optimum location of the transaction log directory for application servers.

   By default, an application server places transaction log files in a subdirectory of the installed WebSphere Application Server, where the subdirectory name is the same as the server name.

   For example, the default directory for an application server named `server1` is `/IBM/WebSphere/AppServer/profiles/`*`profile_name`*`/tranlog/server1`.

   You can define a specific location for the transaction log directory for an application server by setting the **Transaction Log Directory** property for the server. If the directory for the transaction logs has not been created at application server start up, the directory structure is created for you.

   Note: If you change the transaction log directory, you should apply the change and restart the application server as soon as possible, to minimize the risk of problems occurring before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server next starts with the new log directory and is unable to automatically resolve in-flight transactions that were recorded in the old log directory.
4. Never allow more than one application server to concurrently use the same set of log files. Because the transaction logs record the state of global transactions within a server, if the logs become lost or corrupt, then transactions that are in the prepared state before failure can leave resources in an in-doubt state and prevent further updates or access to the resources by other users or servers. These

transactions may need to be manually resolved by either committing or rolling back the transactions at the affected resource managers. The failed server can then be cold-started, which creates new empty transaction logs.

If log files have been mirrored or can be recovered, they can be used when restarting the failed server or moved to an alternate server or machine and another server restarted to perform recovery, as described in the related tasks.

Never allow more than one application server to concurrently use the same set of log files, because each server will destroy the information recorded by the other, resulting in corrupt log files that are unusable for future recovery purposes.

5. Configure application servers to always use the same listening port address at each startup. If you are running distributed transactions between multiple application servers, the remote object references saved in the transaction log need to be redirected to the originating server on recovery.

   On WebSphere Application Server Network Deployment, the node agents automatically redirect such remote object references to the appropriate application servers on recovery. However, if the distributed transaction is between application servers that are not on WebSphere Application Server Network Deployment, then you must handle the redirection of remote object references for transaction recovery to complete. For example, you must do this if an application server is deployed on WebSphere Application Server (not the Network Deployment edition) and runs distributed transactions with non-WebSphere EJB or Corba servers.

   In particular, the default restart action of an application server not on Application Server Network Deployment is to use a different listening port address to the port when the server shut down. This prevents transaction recovery completing. To overcome this, you should always configure application servers to always use the same listening port address at each startup (see the ORB property com.ibm.CORBA.ListenerPort in ORB service settings that can be added to the administrative console). You may need to make similar configuration changes to other application servers involved in transactions, to be able to access those servers during recovery.

Moving a transaction log from one server to another

This topic describes some considerations and actions that you can take to move the transaction logs for an application server to another server.

To move transaction logs from one application server to another, consider the following steps:

1. Move all the transaction log files for the application server. The transaction log directory for each server contains a number of files and subdirectories. When moving transaction logs from one server to another you must move all of the files and subdirectories together as a single unit; otherwise recovery may not complete resulting in data inconsistency.
2. For a server configuration where there are no distributed transactions, move the transaction logs to any server that has access to the same resource managers. For a single server or network-deployed server configuration where it is known there are no distributed transactions present in the logs, the transaction logs can be moved to any server (on any node) that has access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   If the server is in a different cell from the original server, you need to ensure that there is a JAAS alias available to the server that was used by the original server for accessing XA resources. In this case you should use wsadmin to construct the aliases, because if you use the administrative console to create the alias, then the node name gets prefixed to the alias.

   All the transaction log files for the original server need to be moved to a directory accessible by the new server. This can be accomplished by either renaming the transaction log directory or copying all the contents to the new server's transaction log directory before starting the new server.

   Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue

to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

3. For a network-deployed server configuration where there are distributed transactions, move the transaction logs to a server that has the same name and host IP address, and access to the same resource managers. For a network-deployed server configuration, when it is known there are distributed transactions present in the logs, there are more restrictions. Distributed transactions that access multiple servers log information about each server involved in the transaction. This information includes the server name and the IP address of the machine on which the server is running. When recovery is taking place on server restart, the server uses this information to contact the distributed servers and similarly, the distributed servers try to contact the server with the same original name. So, if a server fails and the logs need to the recovered on an alternative server, that alternative server needs to have the same name and host IP address as the original server. The alternative server also needs to have access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   Note: All servers within a cell must have unique names.

   Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

Restarting an application server on a different host

This topic describes some considerations and actions that you can take with transaction logs to restart an application server on a different host.

Moving transactions logs to a different host is similar to moving logs from one server to another, as described in Moving transaction logs from one server to another.

This involves moving an original application server on one host to an alternative server, which has access to the same resource managers, on another host. For a network-deployed server configuration, the alternative server must have the same name and host IP address as the original server.

Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

To restart an application server on a different host, complete the following steps:

1. Ensure that the alternative application server is stopped.
2. Move all the transaction logs for the original server to the alternative application server, according to the considerations described in Moving transaction logs from one server to another.
3. Restart the alternative application server.

Interoperating transactionally between application servers

This topic describes some considerations and actions that you can take to interoperate transactionally between different types of application servers.

WebSphere Application Server is a transaction manager that supports transactional interoperation with other transaction managers through either the CORBA Object Transaction Service (OTS) protocol or, for JSR-109 compliant requests, Web Service Atomic Transaction (WS-AtomicTransaction) protocol. This is in addition to its ability to coordinate XA resource managers and to be coordinated by J2EE Connector 1.5 resource adapters.

- To interoperate transactionally, using the OTS protocol, between WebSphere Application Server Version 6 or later, and versions of the WebSphere Application Server before Version 5.0, you need to set the following system properties on application servers that are before Version 5.0:

  ```
  com.ibm.ejs.jts.jts.ControlSet.nativeOnly=false
  com.ibm.ejs.jts.jts.ControlSet.interoperabilityOnly=true
  ```

  For example, if you want to interoperate between application servers at WebSphere Application Server Version 6 and WebSphere Application Server Version 4.0.n, you need to set the system properties on the Version 4.0.n application servers.
- If you have enabled WebSphere Application Server security, you need to disable protocol security on servers at version 6.0.2 or later to be able to interoperate transactionally with servers at earlier versions.

  To disable protocol security on a server, complete the following steps:

  1. In the administrative console, click **Servers** → **Application Servers** → *server_name***[Container Settings] Container Services** → **Transaction Service**
  2. Clear the **Enable protocol security** check box.
  3. Click **Apply** or **OK**.
  4. Save your changes to the master configuration.
  5. Restart the server.

Configuring an intermediary node for Web services transactions

Intermediary nodes allow the exchange of Web Services Atomic Transaction and Web Services Business Activity protocol messages across firewalls and outside the WebSphere Application Server domain. You configure an intermediary node to specify which WebSphere Application servers the node routes requests to.

1.

2.

You configured the intermediary node ready for use by WebSphere Application Server.

Configure WebSphere Application Server to use the intermediary node by specifying, for each server, the appropriate virtual host of the intermediary node. See “Enabling WebSphere Application Server to use an intermediary node for Web services transactions.”

Enabling WebSphere Application Server to use an intermediary node for Web services transactions

You can use intermediary nodes with Web Services Atomic Transactions or Web Services Business Activities to support the exchange of associated requests across firewalls and outside the WebSphere Application Server domain. You configure WebSphere Application Server to use an intermediary node by specifying the HTTP or HTTPS protocol prefix for the node, in each server that is accessed through the intermediary.

Configure the intermediary node that you want to use, and ensure that you know the address or addresses of the intermediary node that you want to map to servers in your WebSphere Application Server configuration.

1. In the administrative console, click **Servers** → **Application Servers >** *server_name* > **[Container Services] Transaction Service**.
2. Type the prefix for the intermediary node in the **HTTP proxy prefix** or **HTTPS proxy prefix** field. This prefix is the intermediary node address that maps to this server. The HTTPS prefix is used if WebSphere Application Server security is enabled and protocol security is enabled for the transaction service; otherwise the HTTP prefix is used.
3. Save your changes to the master configuration.

4. Repeat the previous steps for each server that is accessed through the intermediary node.
5. Restart the servers.

You configured your system to use an intermediary node. Test your configuration to ensure that messages are routed as you expect.

Configuring a server to use business activity support

Business activity support provides compensation for activities such as sending an e-mail, which can be difficult or impossible to roll back atomically. With this compensation, applications on disparate systems can coordinate activities that are more loosely coupled than atomic transactions. To use the business activity support you must first enable it on each server that you plan to use.

If you have an application component that uses the business activity support, you must enable the support on each server that runs the application.

1. In the administrative console, click **Servers** → **Application servers >** *server_name* **> [Container Settings] Container Services** → **Compensation Service**.
2. Select the **Enable service at server startup** check box.
3. If required, modify the compensation handler retry interval and limit. These values control the frequency with which the CompensationHandler handler compensate and close methods are retried, when either throw a RetryCompensationHandlerException exception, and the number of times that these methods are retried.
4. Save your changes to the master configuration.
5. Repeat the previous steps for each server that you plan to use.
6. Restart all the servers for the changes to take effect.

The business activity support is enabled for the application server. Verify a successful enablement by checking for the message, `CWSCP0005I: The Compensation service started successfully.` in the `SystemOut.log` file for the relevant server.

Deploy the business-activity-enabled application to the server.

Note: Applications can exploit the business activity support only if you deploy them to a WebSphere Application Server Version 6.1 server. Applications cannot use the business activity support if you deploy them to a cluster that includes WebSphere Application Server Version 6.0.x servers.

Chapter 20. Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See Learn about WebSphere applications: Overview and new features for a brief description of each WebSphere extension.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

In addition, now your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java 2 Platform, Enterprise Edition (J2EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application. Read the information about Application extension registry in the *Developing and deploying applications* PDF book.

ActivitySessions

Configuring the default ActivitySession timeout for an application server

Use this task to configure the default ActivitySession timeout for an application server, after which any started ActivitySessions are completed automatically by the ActivitySession service.

The ActivitySession timeout is used to reset any ActivitySession whose remote client has failed to complete the ActivitySession in a timely fashion. The initial default timeout can be configured separately for each application server, and can be overridden programmatically by the setSessionTimeout method of the UserActivitySession interface. If an ActivitySession that contains a transaction reaches the timeout, the transaction's timeout is accelerated so that it is timed out (and rolled back) immediately before the ActivitySession is reset.

To configure the default ActivitySession timeout for an application server, use the WebSphere Administrative console to complete the following steps:

1. Start the WebSphere Administrative console.
2. In the navigation pane, click **Servers** → **Application Servers** This displays a list of the application servers in the content pane.
3. In the Content pane, click the name of the application server that you want to configure. This displays the properties for the application server in the content pane.
4. Under Container Settings, click **Business Process Services** → **ActivitySession Service** This displays the ActivitySession service properties in the content pane.
5. Ensure that the **Enable service at server startup** check box is selected. You must enable the service for the timeout to have an effect.
6. Set the **ActivitySession timeout** property to the default timeout as an integer number of seconds.
   - -1 indicates that ActivitySessions never timeout

- 0 indicates that the default timeout, 300 seconds, applies
- Other values are an integer number of seconds

7. Click **OK**.
8. Save your changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Related concepts

The ActivitySession service
The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. An ActivitySession context can be longer-lived than a global transaction context and can encapsulate global transactions.

ActivitySession service settings

Use this page to configure the properties of the ActivitySession service. The ActivitySession service is a unit-of-work service to coordinate one-phase resources or to extend the activation and passivation of an enterprise bean.

To view this administrative console page, click **Servers** → **Application servers** → *server_name* → **[Container Settings] Business Process Services** → **ActivitySession service**.

Enable service at server startup:

Specifies whether the application server attempts to start the ActivitySession service when the server next starts up.

Default Cleared

Range

Cleared
The server does not try to start the ActivitySession service. If ActivitySessions are to be used in applications that run on this server, the system administrator must select this property then restart the server.

Selected
When the application server starts, it attempts to start the ActivitySession service automatically.

Default timeout:

Specifies the default timeout for an activity session. A server automatically completes an activity session if a remote client has failed to complete the activity session within this time period.

The initial default timeout can be configured separately for each application server, and can be overridden programmatically by the UserActivitySession interface (setSessionTimeout).

Data type Integer

Units Seconds

Default 300 (5 minutes)

Range -1 through 1000000000 seconds
- -1 indicates that ActivitySessions never timeout
- 0 indicates that the default timeout applies
- Other values are an integer number of seconds

Disabling or enabling the ActivitySession service

Use this task to disable or enable the ActivitySession service for an application server.

You can use the ActivitySession **Startup** property to specify whether or not the ActivitySession service is started automatically for an application server.

To disable or enable the ActivitySession service for an application server, use the Administrative console to configure the ActivitySession **Startup** property:

1. Start the Administrative console.
2. In the navigation pane, click **Servers** → **Application Servers** This displays a list of the application servers in the content pane.
3. In the Content pane, click the name of the application server that you want to configure. This displays the properties for the application server in the content pane.
4. Under Container Settings, click **Business Process Services** → **ActivitySession Service** This displays the ActivitySession service properties in the content pane.
5. Select or clear the **Startup** property as needed:
   Selected
   The ActivitySession service is started when the application server is started. This enables applications that specify use of ActivitySessions in their deployment descriptors to run on such an application server.
   Cleared
   [Default] The ActivitySession service is not started when the application server is started. Applications that specify use of ActivitySessions in their deployment descriptors cannot start on such an application server.

   Any attempt to start an application that uses ActivitySessions is rejected and a message issued:

   ```
   WACS0043E: Error found starting an application.  application_name specified an ActivitySession
   attribute that is not allowed when the ActivitySession service is not enabled
   ```

   If this happens during server startup, the server continues to start without the application.
6. Click **OK**.
7. Save your changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

Related concepts

The ActivitySession service
The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. An ActivitySession context can be longer-lived than a global transaction context and can encapsulate global transactions.

Related reference

"ActivitySession service settings" on page 1880
Use this page to configure the properties of the ActivitySession service. The ActivitySession service is a unit-of-work service to coordinate one-phase resources or to extend the activation and passivation of an enterprise bean.

Application profiling

Task overview: Application profiling

You can use application profiling to configure multiple access intent policies on the same entity bean.

Application profiling reflects the fact that different units of work have different use patterns for enlisted entities and can require different kinds of support from the server runtime environment. For more information, see Application Profiling.

1. Assembling applications for application profiles. This topic describes how to configure tasks, create application profiles, and configure tasks on profiles.

2. Managing application profiles. This topic describes how to add and remove tasks from application profiles using the administrative console.
3. Using the TaskNameManager API. This topic describes how to programmatically set the current task name, but you should use this technique sparingly. Wherever possible, use the declarative method instead, which results in more portable function.

Application profiling

You can use application profiling to identify particular units of work to the WebSphere Application Server runtime environment. The run time can tailor its support to the exact requirements of that unit of work.

Access intent is currently the only runtime component that makes use of the application profiling functionality. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Application profiling introduces two new concepts in order to achieve this function: *tasks* and *profiles*.

Tasks A task is a configurable name for a unit of work. *Unit of work* in this case means either a transaction or an ActivitySession. The task name is typically assigned declaratively on a J2EE component that can initiate a unit of work. Most commonly, the task is configured on a method of an Enterprise JavaBeans file that is declared either for container-managed transactions or bean-managed transactions. Any unit of work that begins in the scope of a configured task is associated with that task name. A unit of work can only be named when it is initiated, and the name cannot change for the lifetime of that unit of work. A unit of work ignores any subsequent task name configurations at any point after it has begun. The task is used for the duration of its unit of work to identify configured policies specific to that unit of work.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Profiles

A profile is simply a mapping of a task to a set of access intent policies that are configured on entity beans. When an invocation on a bean (whether by a finder method, a CMR getter, or a dynamic query) requires data to be retrieved from the back end system, the current task associated with the request is used to determine the exact requirement of the transaction. The same bean loads and behaves differently in the context of the task-to-profile mapping. Each profile provides the developer an opportunity to reconfigure the application's access intent. If a request is operating in the absence of a task, the runtime environment uses either a method-level access intent (if any) or a bean-level default access intent.

Note: The application profile configuration is application scope configuration data. If any Enterprise JavaBean (EJB) module contains an application profile configuration, all other EJB modules are implicitly regulated by the Application Profiling service even if they do not contain application profile configuration data.

For example, an application has two EJB modules: EJBModule1 and EJBModule2.

The EJBModule1 has an application profile named AppProfile1. This AppProfile1 is registered by a task named task1. This task1 becomes a *known-to-application task* and is

honored when associated with a unit of work within this application. With the presence of any known-to-application task, method level access intent configurations are ignored and only bean level access intent configurations are applied.

The EJBModule2 contains no application profile configuration data. All entity beans are **not** configured with bean level access intent explicitly, but some methods have method level access intent configurations. If an entity bean in the EJBModule2 is loaded in a unit of work that is associated with task1, the bean-level access intent configuration is applied and method level access intent configuration is ignored. Because the bean level access intent is not set explicitly, the default bean level access intent, which is wsPessimisticUpdate-WeakestLockAtLoad, is applied.

Tasks and units of work considerations:

The application profiling function works under the unit of work (UOW) concept. UOW in this case means either a transaction or an ActivitySession.

The task name on a method is used only when a UOW is begun, because of that method being invoked. This gives it a more predictable data access pattern based on the active unit of work. To be more specific, this approach ensures that a bean type with only one configured access intent is loaded within a UOW, because a bean is configured with only one access intent within an application profile. This configured access intent for a bean type is determined at assembly time and is enforced by the Application Profile service.

A task name is always associated with a unit of work, and that task name does not change for the duration of that UOW. When a UOW associated with a method is begun because of that method being invoked, if a task name is associated with the method then that task name is used to name the UOW. A task assigned to a unit of work is considered a named UOW.

If a task name is not associated with the method that began the UOW, then a default access intent is used and the UOW is unnamed. A unit of work can only be named when the UOW is begun and that task name remains for the life of the UOW. Furthermore, the task assigned to a UOW can never be changed for the life of that UOW. Any task names associated with a method are ignored if that method does not begin a UOW (either container managed or component managed).

It is not possible to change the task name assigned to a unit of work. However, it is possible that in a call sequence consisting of many different application calls a different task name might need to be used for different calls. In this case it is important for the deployer to begin a new UOW and associate with the UOW the necessary task name. For example, assume you have the following beans: sb1 is a session bean, eb2 and eb3 are container managed persistence (CMP) entity beans. When sb1 is called, a transaction is begun and task 't1' is associated with it. Further assume that sb1 then calls eb2 and eb3. If neither eb2 or eb3 create a unit of work, then these beans execute within the UOW context from sb1 and as such its task name (t1). If eb2 or eb3 need to execute within a task name other than t1, then these beans must define a unit of work and associate with it the appropriate task name.

Note that if an application deployer does not specifically configure a transaction on a method, WebSphere Application Server creates a global transaction by default. This is important because if a task is defined on a method, but a UOW is not specifically configured on that method, the EJB container automatically creates a global transaction on behalf of that method. As such, this task name is associated with the UOW and any application profiles mapped to this task are used.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on

requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

Application profiles:

An application profile is the set of access intent policies that should be selectively applied for a particular unit of work (a transaction or *ActivitySession*).

Application profiling enables applications to run under different sets of policies depending on the active task under which the application is operating.

The active task depends upon the current unit of work mechanism. If the current unit of work is a global transaction, then the task is the name associated with that transaction. If the global transaction was not named when it was initiated, then there is no active task anywhere in the scope of that transaction.

If the current unit of work is a local transaction associated with an ActivitySession, then the task is the name associated with that ActivitySession. If the ActivitySession was not named when it was initiated, then there is no active task for any local transaction bound to that ActivitySession. If the current unit of work is a local transaction that is not associated with an ActivitySession, then the task is the name associated with that local transaction. If the local transaction was not associated with a task when the local transaction was initiated, then there is no active task for the duration of that local transaction. In other words, the active task is the task associated with the unit of work on the thread that is coordinating database resources. If the controlling unit of work was not associated with a task when that unit of work was initiated, then there is no active task in the scope of that unit of work.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Consider an application that centralizes the student records for a school district. These records are frequently accessed by the school district's central office in order to generate reports. The report generation process would be optimized if it held no locks with the back end system, and if the records could be read into memory with as few back end operations as possible. Occasionally, however, the records are updated by the students' instructors. Without the ability to distinguish between transactions, the developer is forced to assume a worst-case scenario and, wishing to use pessimistic concurrency, lock the records for all transactions.

Using the application profiling service, the developer can configure in as many ways as necessary the access intent under which the students' records are loaded. Under one profile, the records can be configured with an exclusive pessimistic update intent, not only locking-out competing transactions but ensuring that the student is not removed from the system before the transaction completes. Under another profile, the records can be configured with an optimistic intent as part of an object graph that is read from the back end system in a single database operation. The task represented by the pessimistic profile receives the strong-locking semantics required for certain transactions, while the task represented by the optimistic profile receives the performance benefits appropriate for other transactions.

Application profiling performance considerations:

Application profiling enables assembly configuration techniques that improve your application run time, performance and scalability. You can configure tasks that identify incoming requests, identify access intents determining concurrency and other data access characteristics, and profiles that map the tasks to the access intents.

The capability to configure the application server can improve performance, efficiency and scalability, while reducing development and maintenance costs. The application profiling service has no tuning parameters, other than a checkbox for disabling the service if the service is not necessary. However, the overhead for the application profile service is small and should not be disabled, or unpredictable results can occur.

Access intents enable you to specify data access characteristics. The WebSphere runtime environment uses these hints to optimize the access to the data, by setting the appropriate isolation level and concurrency. Various access intent hints can be grouped together in an access intent policy.

In WebSphere Application Server, it is recommended that you configure bean level access intent for loading a given bean. Application profiling enables you to configure multiple access intent policies on the entity bean, if desired. Some callers can load a bean with the intent to read data, while others can load the bean for update. The capability to configure the application server can improve performance, efficiency, and scalability, while reducing development and maintenance costs.

Access intents enable the EJB container to be configured providing optimal performance based on the specific type of enterprise bean used. Various access intent hints can be specified declaratively at deployment time to indicate to WebSphere resources, such as the container and persistence manager, to provide the appropriate access intent services for every EJB request.

The application profiling service improves overall entity bean performance and throughput by fine tuning the run time behavior. The application profiling service enables EJB optimizations to be customized for multiple user access patterns without resorting to ″worst case″ choices, such as pessimistic update on a bean accessed with the findByPrimaryKey method, regardless of whether the client needs it for read or for an update.

Application profiling provides the capability to define the following hierarchy: **Container-Managed Tasks** > **Application Profiles** > **Access Intent Policies** > **Access Intent Overrides**. Container-managed tasks identify units of work (UOW) and are associated with a method or a set of methods. When a method associated with the task is invoked, the task name is propagated with the request. For example, a UOW refers to a unique path within the application that can correspond to a transaction or ActivitySession. The name of the task is assigned declaratively to a J2EE client or servlet, or to the method of an enterprise bean. The task name identifies the starting point of a call graph or subgraph; the task name flows from the starting point of the graph downstream on all subsequent IIOP requests, identifying each subsequent invocation along the graph as belonging to the configured task. As a best practice, wherever a UOW starts, for example, a transaction or an ActivitySession, assign a task to that starting point.

The application profile service associates the propagated tasks with access intent policies. When a bean is loaded and data is retrieved, the characteristics used for the retrieval of the data are dictated by the application profile. The application profile configures the access intent policy and the overrides that should be used to access data for a specific task.

Access intent policies determine how beans are loaded for specific tasks and how data is accessed during the transaction. The access intent policy is a named group of access intent hints. The hints can be used, depending on the characteristics of the database and resource manager. Various access intent hints applied to the data access operation govern data integrity. The general rule is, the more data integrity, the more overhead. More overhead causes lower throughput and the opportunity for simultaneous data access from multiple clients.

If specified, access intent overrides provide further configuration for the access intent policy.

Best practices

Application profiling is effective in a variety of different scenarios. The following are example situations where application profiling is useful

- **The same bean is loaded with different data access patterns**

  The same bean or set of beans can be reused across applications, but each of those applications has differing requirements for the bean or for beans within the invocation graph. One application can require that beans be loaded for update, while another application requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.
- **Different clients have different data access requirements**

  The same bean or set of beans can be used for different types of client requests. When those clients have different requirements for the bean, or for beans within the invocation graph, application profiling can be used to tailor the bean loading characteristics to the requirements of the client. One client can require beans be loaded for update, while another client requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.

Monitoring tools

You can use the Tivoli Performance Viewer, database and logs as monitoring tools.

You can use the Tivoli Performance Viewer to monitor various metrics associated with beans in an application profiling configuration. The following sections describe at a high level the Tivoli Performance Viewer metrics that reflect changes when access intents and application profiling are used:

- **Collection scope**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor this information to determine the difference between using the ActivitySession scope versus the transaction scope. For the transaction scope, depending on how the container transactions are defined, activates and passivates can be associated with method invocations. The application could use the ActivitySession scope to reduce the frequency of activates and passivates. For more information, see ″Using the ActivitySession service.″
- **Collection increment**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular findByPrimaryKey operation. For example, if the collection increment is set to 10, rather than the default 25, the *Num Activates* value shows 25 for the initial findByPrimaryKey, before any result set iterator runs. If the number of activates rarely exceeds the collection increment, consider reducing the collection increment setting.
- **Resource manager prefetch increment**

  The resource manager prefetch increment is a hint acted upon by the database engine to depend upon the database. The Tivoli Performance Viewer does not have a metric available to show the effect of the resource manager prefetch increment setting.
- **Read ahead hint**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular request. If a read ahead association is not in use, the *Num Activates* value shows a lower initial number. If a read ahead association is in use, the *Num Activates* value represents the number of activates for the entire call graph.

Database tools are helpful in monitoring the different bean loading characteristics that introduce contention and concurrency issues. These issues can be solved by application profiling, or can be made worse by the misapplication of access intent policies.

Database tools are useful for monitoring locking and contention characteristics, such as locks, deadlocks and connections open. For example, for locks the DB2 Snapshot Monitor can show statistics for lock waits, lock time-outs and lock escalations. If excessive lock waits and time-outs are occurring, application profiling can define specific client tasks that require a more string level of locking, and other client tasks that do not require locking. Or, a different access intent policy with less restrictive locking could be applied. After applying this configuration change, the snapshot monitor shows less locking behavior. Refer to information about the database you are using on how to monitor for locking and contention.

The **application server logs** can be monitored for information about rollbacks, deadlocks, and other data access or transaction characteristics that can degrade performance or cause the application to fail.

Application profiling tasks

Tasks are named units of work. They are the mechanism by which the runtime environment determines which access intent policies to apply when an entity bean's data is loaded from the back end system.

Application profiles enable developers to configure an entity bean with multiple access intent policies; if there are *n* instances of profiles in a given application, each bean can be configured with as many as *n* access intent policies.

A task is associated with a transaction or an ActivitySession at the initiation of the unit of work. The task, which cannot change for the lifetime of the unit of work, is always available anywhere within the scope of that unit of work to apply the access intent policy configured for that particular unit of work.

If an enterprise application is configured to use application profiling in any part of the application, then application profiling is active and method-level access intent configurations are ignored when units of works are associated with known-to-application tasks.

If an entity bean is loaded in a unit of work that is not associated with a task, or is associated with a task that is unassociated with an application profile, the default bean-level access intent or the method-level access intent configuration is applied. If a unit of work is associated with a task that is configured with an application profile, the bean-level access intent configuration within the appropriate application profile is applied.

Note: The application profile configuration is application scope configuration data. If any enterprise Javabean (EJB) module contains an application profile configuration, all other EJB modules are implicitly regulated by the Application Profiling service even if they do not contain application profile configuration data.

For example, an application has two EJB modules: EJBModule1 and EJBModule2.

The EJBModule1 has an application profile named AppProfile1. This AppProfile1 is registered by a task named task1. This task1 becomes a *known-to-application task* and is honored when associated with a unit of work within this application. With the presence of any known-to-application task, method level access intent configurations are ignored and only bean level access intent configurations are applied.

The EJBModule2 contains no application profile configuration data. All entity beans are **not** configured with bean level access intent explicitly, but some methods have method level access intent configurations. If an entity bean in the EJBModule2 is loaded in a unit of work that is associated with task1, the bean-level access intent configuration is applied and method level access intent configuration is ignored. Because the bean level access intent is not set explicitly, the default bean level access intent, which is wsPessimisticUpdate-WeakestLockAtLoad, is applied.

The active task depends upon the current unit of work mechanism. If the current unit of work is a global transaction, then the task is the name associated with that transaction. If the global transaction was not named when it was initiated, then there is no active task anywhere in the scope of that transaction.

If the current unit of work is a local transaction associated with an ActivitySession, then the task is the name associated with that ActivitySession. If the ActivitySession was not named when it was initiated, then there is no active task for any local transaction bound to that ActivitySession. If the current unit of work is a local transaction that is not associated with an ActivitySession, then the task is the name associated with that local transaction. If the local transaction was not associated with a task when the local transaction was initiated, then there is no active task for the duration of that local transaction. In other words, the active task is the task associated with the unit of work on the thread that is coordinating database resources. If the controlling unit of work was not associated with a task when that unit of work was initiated, then there is no active task in the scope of that unit of work.

For example, consider a school district application that calls through a session bean in order to interact with student records. One method on the session bean allows administrators to modify the students' records; another method supports student requests to view their own records. Without application profiling, the two tasks would operate anonymously and the runtime environment would be unable to distinguish work operating on behalf of one task or the other. To optimize the application, a developer can configure one of the methods on the session bean with the task ″updateRecords″ and the other method on the session bean with the task ″readRecords″. When registered with an application profile that has the student bean configured with the appropriate locking access intent, the ″updateRecords″ task is assured that it is not unnecessarily blocking transactions that need to only read the records. For more information about the relationships between tasks and units of work, see “Tasks and units of work considerations” on page 1883.

Tasks can be configured to be managed by the container or to be programmatically established by the application. Container managed tasks can be configured on servlets, JavaServer Pages (JSP) files, application clients, and the methods of Enterprise JavaBeans (EJB). Configured container-managed tasks are only associated with units of work that the container initiates after the task name is set. Application managed tasks can be configured on all J2EE components. In the case of enterprise beans they must be bean managed transactions.″

best-practices: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Application profiling interoperability

Using application profiling with 5.x compatibility mode or in a clustered environment with mixed WebSphere Application Server product versions and mixed platforms can affect its behavior in different ways.

The effect of 5.x Compatibility Mode

Application profiling supports *forward* compatibility. Application profiles created in previous versions of WebSphere Application Server (Enterprise Edition 5.0 or WebSphere Business Integration Server Foundation 5.1) can only run in WebSphere Application Server Version 6 or later if the Application Profiling 5.x Compatibility Mode attribute is turned on. If the 5.x Compatibility Mode attribute is off, Version 5 application profiles might display unexpected behavior.

Similarly, application profiles that you create using the latest version of WebSphere Application Server are not compatible with Version 5 or earlier versions. Even applications configured with application profiles run

on Version 6.x servers with the Application Profiling 5.x Compatibility Mode attribute turned on cannot interact with applications configured with profiles run on Version 5 servers.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to **true** in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the *launchClient* command.

Considerations for a clustered environment

In a clustered environment with mixed WebSphere Application Server product versions and mixed platforms, applications configured with application profiles might exhibit unexpected behavior because previous versions of server members cannot support the application profiling of Version 6.x.

If a clustered environment contains both Version 5.x and 6.x server members, and if any applications are configured with application profiles, the Application Profiling 5.x Compatibility Mode attribute must be turned on in Version 6 server members. Still, this cluster can only support Version 5 application profiling behavior. To support applications configured with Version 6 application profiles in a cluster environment, all server members in the cluster must be at the Version 6.x level.

WebSphere Application Server Enterprise Edition Version 5.0.2

If you use WebSphere Application Server Enterprise Edition 5.0.2, you must apply WebSphere Application Server Version 5 service pack 7 or later service pack to enable Application Profiling interoperability.

Managing application profiles

Using the administrative console, you can add tasks to or remove tasks from application profiles.

1. Start the administrative console.
2. Select **Applications > Enterprise Applications >** *application_name* **> Application Profiles >** *profile_name* **> Tasks**.
3. On the Tasks collection page, you can add new tasks to the profile, delete tasks, edit current task settings, and so on.

   Note that, within the scope of an application, no task can be configured on more than one application profile. In such a situation, your application cannot be restarted until you correct the configuration.
4. Save your configuration.
5. Restart the application in order for your changes to take effect.

Using the TaskNameManager interface

Using the TaskNameManager interface, you can programmatically set the current task name. It enables both overriding of the current task associated with the thread of execution and resetting of the current task with the original task.

Except for J2EE 1.3 applications that are running on a server where the 5.x Compatibility Mode attribute is selected, this interface cannot be used within Enterprise JavaBeans that are configured for container-managed transactions or container-managed ActivitySessions because units of work can only be associated with a task at the exact time that the unit of work is initiated. The call to set the task name must therefore be invoked before the unit of work is begun. Units of work cannot be named after they are begun. Calls on this interface during the execution of a container-managed unit of work are simply ignored.

Application profiling does not support queries of the task that is in operation at run time. Instead, applications interact with logical task names that are declaratively configured as application managed tasks. Logical references enable the actual task name to be changed without having to recompile applications.

Wherever possible, avoid setting tasks programmatically. The declarative method results in more portable function that can be easily adjusted without requiring redevelopment and recompilation.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.0 client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

1. Configure application-managed tasks. Application profiling requires that a task name reference be declared for any task that is to be set programmatically. Task name references introduce a level of indirection so that the actual task set at run time can be adjusted by reassembly without requiring recoding or recompilation. Any attempt to set a task name that is undeclared as a task reference results in the raising of an exception. If a unit of work has already begun when a task name is set, then that existing unit of work is not associated with the task name. Only units of work that are begun after the task name is set are associated with the task.

   Configure application-managed tasks as described in the following topics:
   - Configuring application managed tasks for web components.
   - Configuring application managed tasks for application clients.
   - Configuring application managed tasks for Enterprise JavaBeans.
2. Perform a Java Naming and Directory Interface (JNDI) lookup on the TaskNameManager interface:

   ```
   InitialContext ic = new InitialContext();
   TaskNameManager tnManager = ic.lookup
   ("java:comp/websphere/AppProfile/TaskNameManager");
   ```

   The *TaskNameManager* interface is not bound into the namespace if the application profiling service is disabled.
3. Set the task name:

   ```
   try {
   tnManager.setTaskName("updateAccount");
   }
   catch (IllegalTaskNameException e) {
   // task name reference not configured. Handle error.
   }
   // . . .
   //
   ```

   The name passed to the setTaskName() method (″updateAccount″ in this example) is actually a task name reference that you configured in step one.
4. Begin a UserTransaction

   Note: If you are using a J2EE 1.3 application with the 5.x Compatibility mode set, the task name set in step 3 is now the active task name and you can disregard this step.

   If you are using a J2EE application and the compatibility mode is not set, or if you are using a J2EE 1.4 application, you must begin a transaction for the task name to become active. A task name can

only be associated with a transaction. Furthermore, it is associated with a transaction when that transaction is begun, and that task name is associated with the transaction for the life of the transaction. Therefore, the task name set above is not active at this point. You must begin a UserTransaction as the following code snippet illustrates:

```
try{
    InitialContext initCtx = new InitialContext();
    userTran = (UserTransaction) initCtx.lookup("java:comp/UserTransaction");
    userTran.begin();
}
catch(Exception e){
}
// . . .
//
```

Notice the resetTaskName() method on the TaskNameManager interface. Resetting the task name has no effect unless called by a J2EE 1.3 application running on a server for which the 5.x Compatibility Mode attribute is selected on the Application Profile Service's console page. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. A call to resetTask( ) should only be used by J2EE 1.3 applications when the 5.x Compatibility mode is set to undo the effects of any setTaskName() method operations and reestablish whatever task name was current when the component began execution. If the setTaskName() method has not been called, the resetTaskName() method has no effect.

TaskNameManager interface:

The TaskNameManager is the programmatic interface to the application profiling function. Because on rare occasions it may be necessary to programmatically set the current task name, the TaskNameManager interface enables both overriding of the current task associated with the thread of execution and resetting of the current task with the original task.

Application profiling enables you to identify particular units of work to the WebSphere Application Server runtime environment. The run time can tailor its support to the exact requirements of that unit of work. Access intent is currently the only runtime component that makes use of the application profiling functionality. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Application profiling introduces two concepts in order to achieve this function: tasks and profiles.

A *task* is a configurable name for a unit of work. *Unit of work* in this case means either a transaction or an ActivitySession.

A *profile* is simply a mapping of a task to a set of access intent policies that are configured on entity beans. When an invocation on a bean (whether by a finder method, a container managed relationship (CMR) getter, or a dynamic query) requires data to be retrieved from the back end system, the task of the active unit of work associated with the request is used to determine the exact requirement of the transaction. The same bean loads and behaves differently in the context of the task-to-profile mapping. Each profile provides the developer an opportunity to reconfigure the application's access intent.

Except for J2EE 1.3 applications that are executing on a server where the *5.x Compatibility Mode* attribute is selected, this interface cannot be used within Enterprise JavaBeans that are configured for container-managed transactions or container-managed ActivitySessions because units of work can only be associated with a task at the exact time that the unit of work is initiated. The call to set the task name must therefore be started before the unit of work is begun. Units of work cannot be named after they are begun. Calls on this interface during the execution of a container-managed unit of work are simply ignored.

The TaskNameManager interface is available to all J2EE components using the following Java Naming and Directory Interface (JNDI) lookup:

```
java:comp/websphere/AppProfile/TaskNameManager
package com.ibm.websphere.appprofile;

/**
* The TaskNameManager is the programmatic interface
* to the application profiling function. Using this interface,
* programmers can set the current task name on the
* thread of execution. The task name must have been
* configured in the deployment descriptors as a task
* reference associated with a task. The set task
* name's scope is the duration of the method
* invocation in the EJB and Web components and for
* the duration of the client process, or until the
* resetTaskName() method is invoked.
*/
public interface TaskNameManager {

/**
* Set the thread's current task name to the specified
* parameter. The task name must have been configured as
* a task reference with a corresponding task or the
* IllegalTaskName exception is thrown.
*/
public void setTaskName(String taskName) throws IllegalTaskNameException;

/**
* Sets the thread's task name to the value that was set
* at, or imported into, the beginning of the method
* invocation (for EJB and Web components) or process
* (for J2EE clients).
*/
public void resetTaskName();

}
```

Application profiling exceptions

The following exceptions are thrown in response to various illegal actions related to application profiling.

com.ibm.ws.exception.RuntimeWarning

This exception is thrown when the application is started, if the application is configured incorrectly. The startup is consequently terminated. Some examples of bad configurations include:

- A task configured on two different application profiles.
- A method configured with two different task run-as policies .

com.ibm.websphere.appprofile.IllegalTaskNameException

This exception is raised if an application attempts to programmatically set a task when that task has not been configured as a task name reference.

Application profiling service settings

Use this page to enable or disable the application profiling service.

Applications that are configured to use the application profiling service do not start successfully unless the application profiling service is enabled.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Container Services> Application Profiling Service**.

Enable service at server startup:

Specifies whether the server attempts to start the application profiling service.

Default	Selected

Range

Selected
When the application server starts, it attempts to start the application profiling service automatically.

Cleared
The application profiling service is not enabled when an application server starts. Applications configured with application profiling cannot be started on servers that do not enable the application profiling service.

5.x Compatibility Mode:

When selected, J2EE 1.3 applications that use application profiling execute exactly as they did in the 5.x releases of WebSphere Application Server.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the *launchClient* command.

Operation in this mode can lead to unexpected deadlocks during database access. Also, tasks do not propagate on remote invocations between J2EE 1.3 and J2EE 1.4 applications, possibly resulting in the use of unexpected access intent policies. This mode also results in performance degradation if applications configured with application profiling are installed on the server.

Support for J2EE 1.3 applications operating with 5.x Compatibility Mode = true is deprecated as of WebSphere Application Server Version 6.x. When cleared, J2EE 1.3 applications that use application profiling execute with the same constraints as J2EE 1.4 applications. In this mode, tasks are established only when a new unit of work begins. This means the complete unit of work executes under at most one task.

Default
Selected

Range

Selected
J2EE 1.3 applications that are dependent on the behavior of application profiling service for Version 5.x can run with the same behavior in Version 6.0.

Cleared
Tasks are established only when a new global unit of work begins.

Application profile collection

Use this page to view application profiles and manage tasks associated with application profiles.

An application profile is a set of policies that are to be applied during the execution of an enterprise bean and a set of tasks that are associated with that profile. Mapping tasks to application profiles will control which access intent policies are applied at run time for the units of work that correspond to a particular task.

You can use an assembly tool such as Application Server Toolkit (AST) or Rational Application Developer to add or delete application profiles. The AST is shipped with WebSphere Application Server version 6.x.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Application Profiles**.

Name:

The name of the application profile.

The name must be unique; multiple profiles cannot share the same name.

Data type String

Description:

A description of the application profile.

Data type String

Application profile settings:

Use this page to modify application profile settings.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Application Profiles >** *application_profile_name*.

Name:

The name of the application profile.

The name must be unique; multiple profiles cannot share the same name.

Data type String

Description:

A description of the application profile.

Data type String

Task collection:

Use this page to manage tasks.

Requests associated with any of the configured tasks operate under the access-intent policies that are configured with the profile. A task can be configured on only *one* application profile.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Application Profiles >** *application_profile_name* **> Tasks**.

Name:

The name of the task.

The task name must be unique among the set of application profiles.

Data type String

Description:

A description of the task.

Data type String

Task settings:

Use this page to modify task settings.

To view this administrative console page, click **Applications > Enterprise Applications >** *application_name* **> Application Profiles >** *application_profile_name* **> Tasks >** *task_name*.

Name:

The name of the task.

The task name must be unique among the set of application profiles.

Data type String

Description:

A description of the task.

Data type String

Asynchronous beans

Using asynchronous beans

The asynchronous beans feature adds a new set of APIs that enable Java 2 Platform Enterprise Edition J2EE applications to run asynchronously inside an Integration Server.

This topic provides a brief overview of the tasks involved in using asynchronous beans. For a more detailed description of the asynchronous beans model, review the conceptual topic Asynchronous beans. For detailed information on the programming model for supported asynchronous beans interfaces, see the topic Work managers.

1. Configure work managers.
2. Configure timer managers.
3. Assemble applications that use asynchronous beans work managers.
4. Develop work objects to run code in parallel.
5. Develop event listeners.
6. Develop asynchronous scopes.

Asynchronous beans

An asynchronous bean is a Java object or enterprise bean that can run asynchronously by a Java 2 Platform Enterprise Edition (J2EE) application, using the J2EE context of the asynchronous bean creator.

Asynchronous beans can improve performance by enabling a J2EE program to decompose operations into parallel tasks. Asynchronous beans support the construction of stateful, active J2EE applications. These applications address a segment of the application space that J2EE has not previously addressed (that is, advanced applications that require application threading, active agents within a server application, or distributed monitoring capabilities).

Asynchronous beans can run using the J2EE security context of the creator J2EE component. These beans also can run with copies of other J2EE contexts, such as:

- Internationalization context
- Application profiles, which are not supported for J2EE 1.4 applications and deprecated for J2EE 1.3 applications
- Work areas

Asynchronous bean interfaces

Four types of asynchronous beans exist:

Work object

There are two work interfaces that essentially accomplish the same goal. The legacy Asynchronous Beans work interface is com.ibm.websphere.asynchbeans.Work, and the CommonJ work interface is commonj.work.Work. A work object runs parallel to its caller using the work manager `startWork` or `schedule` method (`startWork` for legacy Asynchronous Beans and `schedule` for CommonJ). Applications implement work objects to run code blocks asynchronously. For more information on the Work interface, see the API documentation.

Timer listener

This interface is an object that implements the commonj\timers\TimerListener interface. Timer listeners are called when a high-speed transient timer expires. For more information on the TimerListener interface, see the API documentation.

Alarm listener

An alarm listener is an object that implements the com.ibm.websphere.asynchbeans.AlarmListener interface. Alarm listeners are called when a high-speed transient alarm expires. For more information on the AlarmListener interface, see the API documentation.

Event listener

An event listener can implement any interface. An event listener is a lightweight, asynchronous notification mechanism for asynchronous events within a single Java virtual machine (JVM). An event listener typically enables J2EE components within a single application to notify each other about various asynchronous events.

Supporting interfaces

Work manager

Work managers are thread pools that administrators create for J2EE applications. The administrator specifies the properties of the thread pool and a policy that determines which J2EE contexts the asynchronous bean inherits.

CommonJ Work manager

The CommonJ work manager is similar to the work manager. The difference between the two is that the CommonJ work manager contains a subset of the asynchronous beans work manager methods. Although CommonJ work manager functions in a J2EE 1.4 environment, each JNDI lookup of a work manager does not return a new instance of the WorkManager. All the JNDI lookup of work managers within a scope have the same instance.

Timer manager

Timer managers implement the commonj.timers.TimerManager interface, which enables J2EE applications, including servlets, EJB applications, and JCA Resource Adapters, to schedule future timer notifications and receive timer notifications. The timer manager for Application Servers specification provides an application-server supported alternative to using the J2SE `java.util.Timer` class, which is inappropriate for managed environments.

Event source

An event source implements the com.ibm.websphere.asynchbeans.EventSource interface. An event source is a system-provided object that supports a generic, type-safe asynchronous notification server within a single JVM. The event source enables event listener objects, which implement any interface to be registered. For more information on the EventSource interface, see the API documentation.

Event source events

Every event source can generate its own events, such as listener count changed. An application

can register an event listener object that implements the class com.ibm.websphere.asynchbeans.EventSourceEvents. This action enables the application to catch events such as listeners being added or removed, or a listener throwing an unexpected exception. For more information on the EventSourceEvents class, see the API documentation.

Additional interfaces, including alarms and subsystem monitors, are introduced in the topic Developing Asynchronous scopes, which discusses some of the advanced applications of asynchronous beans.

Transactions

Every asynchronous bean method is called using its own transaction, much like container-managed transactions in typical enterprise beans. It is very similar to the situation when an Enterprise Java Beans (EJB) method is called with TX_NOT_SUPPORTED. The runtime starts a local transaction before invoking the method. The asynchronous bean method is free to start its own global transaction if this transaction is possible for the calling J2EE component. For example, if an enterprise bean creates the component, the method that creates the asynchronous bean must be TX_BEAN_MANAGED.

When you call an entity bean from within an asynchronous bean, for example, you must have a global transactional context available on the current thread. Because asynchronous bean objects start local transactional contexts, you can encapsulate all entity bean logic in a session bean that has a method marked as TX_REQUIRES or equivalent. This process establishes a global transactional context from which you can access one or more entity bean methods.

If the asynchronous bean method throws an exception, any local transactions are rolled back. If the method returns normally, any incomplete local transactions are completed according to the unresolved action policy configured for the bean. EJB methods can configure this policy using their deployment descriptor. If the asynchronous bean method starts its own global transaction and does not commit this global transaction, the transaction is rolled back when the method returns.

Access to J2EE component metadata

If an asynchronous bean is a J2EE component, such as a session bean, its own metadata is active when a method is called. If an asynchronous bean is a simple Java object, the J2EE component metadata of the creating component is available to the bean. Like its creator, the asynchronous bean can look up the java:comp namespace. This look up enables the bean to access connection factories and enterprise beans, just as it would if it were any other J2EE component. The environment properties of the creating component also are available to the asynchronous bean.

The `java:comp` namespace is identical to the one available for the creating component; the same restrictions apply. For example, if the enterprise bean or servlet has an EJB reference of `java:comp/env/ejb/MyEJB`, this EJB reference is available to the asynchronous bean. In addition, all of the connection factories use the same resource-sharing scope as the creating component.

Connection management

An asynchronous bean method can use the connections that its creating J2EE component obtained using java:comp resource references. (For more information on resource references, see References). However, the bean method must access those connections using a get, use or close pattern. There is no connection caching between method calls on an asynchronous bean. The connection factories or datasources can be cached, but the connections must be retrieved on every method call, used, and then closed. While the asynchronous bean method can look up connection factories using a global Java Naming and Directory Interface (JNDI) name, this is not recommended for the following reasons:

- The JNDI name is hard coded in the application (for example, as a property or string literal).
- The connection factories are not shared because there is no way to specify a sharing scope.

For code examples that demonstrate both the correct and the incorrect ways to access connections from asynchronous bean methods, see the topic Example: Asynchronous bean connection management.

Deferred start of Asynchronous Beans

Asynchronous beans support deferred start by allowing serialization of J2EE service context information. The `WorkWithExecutionContext createWorkWithExecutionContext(Work r)` method on the WorkManager interface will create a snapshot of the J2EE service contexts enabled on the WorkManager. The resulting `WorkWithExecutionContext` object can then be serialized and stored in a database or file. This is useful when it is necessary to store J2EE service contexts such as the current security identity or Locale and later inflate them and run some work within this context. The `WorkWithExecutionContext` object can run using the startWork() and doWork() methods on the WorkManager interface.

All `WorkWithExecutionContext` objects must be deserialized by the same application that serialized it. All EJBs and classes must be present in order for Java to successfully inflate the objects contained within.

Deferred start and security

The asynchronous beans security service context might require Common Secure Interoperability Version 2 (CSIv2) identity assertion to be enabled. Identity assertion is required when a `WorkWithExecutionContext` object is deserialized and run to Java Authentication and Authorization Service (JAAS) subject identity credential assignment. Review the following topics to better understand if you need to enable identity assertion, when using a `WorkWithExecutionContext` object:

- Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocol
- Identity Assertion

There are also issues with interoperating with `WorkWithExecutionContext` objects from different versions of the product. See Interoperating with asynchronous beans .

Work managers:

A work manager is a thread pool created for J2EE applications that use asynchronous beans.

Using the administrative console, an administrator can configure any number of work managers. The administrator specifies the properties of the work manager, including the J2EE context inheritance policy for any asynchronous beans that use the work manager. The administrator binds each work manager to a unique place in Java Naming and Directory Interface (JNDI). You can use work manager objects in any one of the following interfaces:

- Asynchronous beans
- CommonJ work manager (For details, see the CommonJ work manager section in this article.)

The selected type of interface is resolved during the JNDI lookup time. The interface type is the value that you specify in the ResourceRef, rather than the interface type specified in the configuration object. For example, you can have one ResourceRef for each interface per configuration object, and each ResourceRef lookup returns that appropriate type of instance.

The work managers provide a programming model for the J2EE 1.4 applications. For more information, see the Programming model section in this article.

When writing a Web or EJB component that uses asynchronous beans, the developer should include a resource reference in each component that needs access to a work manager. For more information on resource references, see the article References. The component looks up a work manager using a logical name in the component, java:comp namespace, just as it looks up a data source, enterprise bean or connection factory.

The deployer binds physical work managers to logical work managers when the application is deployed.

For example, if a developer needs three thread pools to partition work between bronze, silver, and gold levels, the developer writes the component to pick a logical pool based on an attribute in the client application profile. The deployer has the flexibility to decide how to map this request for three thread pools. The deployer might decide to use a single thread pool on a small machine. In this case, the deployer binds all three resource references to the same work manager instance (that is, the same JNDI name). A larger machine might support three thread pools, so the deployer binds each resource reference to a different work manager. Work managers can be shared between multiple J2EE applications installed on the same server.

An application developer can use as many logical work managers as necessary. The deployer chooses whether to map one physical work manager or several to the logical work manager defined in the application.

All J2EE components that need to share asynchronous scope objects must use the same work manager. These scope objects have an affinity with a single work manager. An application that uses asynchronous scopes should verify that all of the components using scope objects use the same work manager.

When multiple work managers are defined, the underlying thread pools are created in a Java virtual machine (JVM) only if an application within that JVM looks up the work manager. For example, there might be ten thread pools (work managers) defined, but none are actually created until an application looks these pools up.

Note: Asynchronous beans do not support submitting work to remote JVMs.

CommonJ Work Manager

The CommonJ work manager is similar to the work manager. The difference between the two is that the CommonJ work manager contains a subset of the asynchronous beans work manager methods. Although CommonJ work manager functions in a J2EE 1.4 environment, the interface does not return a new instance for each JNDI naming lookup, since this specification is not included in the J2EE specification.

Remote start of work. The CommonJ Work specification optional feature for work running remotely is not supported. Even if a unit of work implements the `java.io.Serializable` interface, the unit of work does not run remotely.

How to look up a work manager

An application can look up a work manager as follows. Here, the component contains a resource reference named `wm/myWorkManager`, which was bound to a physical work manager when the component was deployed:

```
InitialContext ic = new InitialContext();
WorkManager wm = (WorkManager)ic.lookup("java:comp/env/wm/myWorkManager");
```

Inheritance J2EE contexts

Asynchronous beans can inherit the following J2EE contexts.

Internationalization context
: When this option is selected and the internationalization service is enabled, and the internationalization context that exists on the scheduling thread is available on the target thread.

Work area
: When this option is selected, the work area context for every work area partition that exists on the scheduling thread is available on the target thread.

Application profile (deprecated)
: Application profile context is not supported and not available for J2EE 1.4 applications. For J2EE

1.3 applications, when this option is selected, the application profile service is enabled, and the application profile service property, **5.x compatibility mode**, is selected. The application profile task that is associated with the scheduling thread is available on the target thread for J2EE 1.3 applications. For J2EE 1.4 applications, the application profile task is a property of its associated unit of work, rather than a thread. This option has no effect on the behavior of the task in J2EE 1.4 applications. The scheduled work that runs in a J2EE 1.4 application does not receive the application profiling task of the scheduling thread.

Security

The asynchronous bean can be run as anonymous or as the client authenticated on the thread that created it. This behavior is useful because the asynchronous bean can do only what the caller can do. This action is more useful than a RUN_AS mechanism, for example, which prevents this kind of behavior. When you select the **Security** option, the JAAS subject that exists on the scheduling thread is available on the target thread. If not selected, the thread runs anonymously.

Component metadata

Component metadata is relevant only when the asynchronous bean is a simple Java object. If the bean is a J2EE component, such as an enterprise bean, the component metadata is active.

The contexts that can be inherited depend on the work manager used by the application that creates the asynchronous bean. Using the administrative console, the administrator defines the sticky context policy of a work manager by selecting the services on which the work manager is to be made available.

Programming model

Work managers support the following programming models.

- **CommonJ Specification**. The Application Server Version 6.0 CommonJ programming model uses the WorkManager and TimerManager to manage threads and timers asynchronously in the J2EE 1.4 environment.
- **Asynchronous beans and CommonJ specification extensions**. The current asynchronous beans Event Source, asynchronous scopes, subsystem monitors and J2EEContext interfaces are a part of the CommonJ extension.

The following table describes the method mapping between the CommonJ and Asynchronous beans APIs. You can change the current asynchronous beans interfaces to use the CommonJ interface, while maintaining the same functions.

CommonJ package	API	Asynchronous beans package	API
Work manager		Work manager	
Asynchronous beans	Field - IMMEDIATE (long)		Field - IMMEDIATE (int)
	Field - INDEFINITE		Field - INDEFINITE
	schedule(Work) throws WorkException, IllegalArgumentException		startWork(Work) throws WorkException, IllegalArgumentException
	schedule(Work, WorkListener) throws WorkException, IllegalArgumentException **Note:** Configure the work manager work timeout property to the value you previously specified as `timeout_ms` on `startWork`. The default timeout value is INDEFINITE.		startWork(Work, timeout_ms, WorkListener) throws WorkException, IllegalArgumentException

	waitForAll(workItems, timeout_ms)		join(workItems, JOIN_AND, timeout_ms)
	waitForAny(workItems, timeout_ms)		join(workItems, JOIN_OR, timeout_ms)
WorkItem		WorkItem	
	getResult		getResult
	getStatus		getStatus
WorkListener		WorkListener	
	workAccepted(WorkEvent)		workAccepted(WorkEvent)
	workCompleted(WorkEvent)		workCompleted(WorkEvent)
	workRejected(WorkEvent)		workRejected(WorkEvent)
	workStarted(WorkEvent)		workStarted(WorkEvent)
WorkEvent		WorkEvent	
	Field - WORK_ACCEPTED		Field - WORK_ACCEPTED
	Field - WORK_COMPLETED		Field - WORK_COMPLETED
	Field - WORK_REJECTED		Field - WORK_REJECTED
	Field - WORK_STARTED		Field - WORK_STARTED
	getException		getException
	getType		getType
	getWorkItem().getResult() **Note:** This API is valid only after the work is complete.		getWork
Work	(extends Runnable)	Work	(Extends Runnable)
	isDaemon		*
	release		release
RemoteWorkItem	Not in this release. Use Distributed WorkManager in Extended Deployment or future release	NA	
TimerManager		AlarmManager	
	resume		*
	schedule(Listener, Date)		create(Listener, context, time) ** need to convert the parameters
	schedule(Listener, Date, period)		
	schedule(Listener, delay, period)		
	scheduleAtFixedRate(Listener, Date, period)		
	scheduleAtFixedRate(Listener, delay, period)		
	stop		
	suspend		
Timer		Alarm	
	cancel		cancel

	getPeriod		
	getTimerListener		getAlarmListener
	scheduledExecutionTime		
TimerListener		AlarmListener	
	timerExpired(timer)		fired(alarm)
StopTimerListener		Not applicable	
	timerStop(timer)		
CancelTimerListener		Not applicable	
	timerCancel(timer)		
WorkException	(Extends Exception)	WorkException	(Extends WsException)
WorkCompletedException	(Extends WorkException)	WorkCompletedException	(Extends WorkException)
WorkRejectedException	(Extends WorkException)	WorkRejectedException	(Extends WorkException)

For more information on work manager APIs, refer to the Javadoc.

Work manager examples

Table 55. Look up work manager

Asynchronous beans	CommonJ
`InitialContext ctx = new InitialContext();` `com.ibm.websphere.asynchbeans.WorkManager wm =` `(com.ibm.websphere.asynchbeans.WorkManager)` `ctx.lookup("java:comp/env/wm/MyWorkMgr");`	`InitialContext ctx = new InitialContext();` `commonj.work.WorkManager wm =` `(commonj.work.WorkManager)` `ctx.lookup("java:comp/env/wm/MyWorkMgr");`

Table 56. Create your work using MyWork

Asynchronous beans	CommonJ
`public class MyWork implements` `com.ibm.websphere.asynchbeans.Work {` `public void release() {` `.....` `}` `public void run() {` `System.out.println("Running.....");` `}`	`public class MyWork implements` `commonj.work.Work{` `public boolean isDaemon() {` `return false;` `}` `public void release () {` `.....` `}` `public void run () {` `System.out.println("Running.....");` `}`

Table 57. Submit the work

Asynchronous beans	CommonJ

Table 57. Submit the work (continued)

MyWork work1 = new MyWork(new URI = "http://www.example./com/1"); MyWork work2 = new MyWork(new URI = "http://www.example./com/2"); WorkItem item1; WorkItem item2; Item1=wm.startWork(work1); Item2=wm.startWork(work2); // case 1: block until all items are done ArrayList col1 = new ArrayList(); Col1.add(item1); Col1.add(item2); wm.join(col1, WorkManager.JOIN_AND, (long)WorkManager.IMMEDIATE); // when the works are done System.out.println("work1 data="+work1.getData()); System.out.println("work2 data="+work2.getData()); // case 2: wait for any of the items to complete. Boolean ret = wm.join(col1, WorkManager.JOIN_OR, 1000);	MyWork work1 = new MyWork(new URI = "http://www.example./com/1"); MyWork work2 = new MyWork(new URI = "http://www.example./com/2"); WorkItem item1; WorkItem item2; Item1=wm.schedule(work1); Item2=wm.schedule(work2); // case 1: block until all items are done Collection col1 = new ArrayList(); col1.add(item1); col1.add(item2); wm.waitForAll(col1, WorkManager.IMMEDIATE); // when the works are done System.out.println("work1 data="+work1.getData()); System.out.println("work2 data="+work2.getData()); // case 2: wait for any of the items to complete. Collection finished = wm.waitForAny(col1, // check the workItems status if (finished != null) { Iterator I = finished.iterator(); if (i.hasNext()) { WorkItem wi = (WorkItem) i.next(); if (wi.equals(item1)) { System.out.println("work1 = "+ work1.getData()); } else if (wi.equals(item2)) { System.out.println("work1 = "+ work1.getData()); } } }

Table 58. Create a timer manager

Asynchronous beans	CommonJ
InitialContext ctx = new InitialContext(); com.ibm.websphere.asynchbeans.WorkManager wm = (com.ibm.websphere.asynchbeans.WorkManager) ctx.lookup("java:comp/env/wm/MyWorkMgr"); AsynchScope ascope; Try { Ascope = wm.createAsynchScope("ABScope"); } Catch (DuplicateKeyException ex) { Ascope = wm.findAsynchScope("ABScope"); ex.printStackTrace(); } // get an AlarmManager AlarmManager aMgr= ascope.getAlarmManager();	InitialContext ctx = new InitialContext(); Commonj.timers.TimerManager tm = (commonj.timers.TimerManager) ctx.lookup("java:comp/env/tm/MyTimerManager");

Table 59. Fire the timer

Asynchronous beans	CommonJ

Table 59. Fire the timer (continued)

`// create alarm` `ABAlarmListener listener = new ABAlarmListener();` `Alarm am =` `  aMgr.create(listener, "SomeContext", 1000*60);`	`// create Timer` `TimerListener listener =` `new StockQuoteTimerListener("qqq",` `"johndoe@example.com");` `Timer timer = tm.schedule(listener, 1000*60);` `// Fixed-delay: schedule timer to expire in` `// 60 seconds from now and repeat every` `// hour thereafter.` `Timer timer = tm.schedule(listener, 1000*60,` `1000*30);` `// Fixed-rate: schedule timer to expire in` `// 60 seconds from now and repeat every` `// hour thereafter` `Timer timer = tm.scheduleAtFixedRate(listener,` `1000*60, 1000*30);`

Timer managers:

The timer manager combines the functions of the asynchronous beans alarm manager and asynchronous scope. So, when a timer manager is created, it internally uses an asynchronous scope to provide the timer manager life cycle functions.

You can look up the timer manager in the JNDI name space. This capability is different from the alarm manager that is retrieved through the asynchronous beans scope. Each lookup of the timer manager returns a new logical timer manager that can be destroyed independently of all other timer managers.

A timer manager can be configured with a number of thread pools through the administrative console. For deployment you can bind this timer manager to a resource reference at assembly time, so the resource reference can be used by the application to look up the timer manager.

The Java code to look up the timer manager is:

```
InitialContext ic = new InitialContext();
TimerManager tm = (TimerManager)ic.lookup("java:comp/env/tm/TimerManager");
```

The programming model for setting up the alarm listener and the timer listener is different. The following code example shows that difference.

Table 60. Set up the timer listener

Asynchronous beans	**CommonJ**

Table 60. Set up the timer listener (continued)

public class ABAlarmListener implements AlarmListener { public void fired(Alarm alarm) { System.out.println("Alarm fired. Context =" + alarm.getContext()); }	public class StockQuoteTimerListener implements TimerListener { String context; String url; public StockQuoteTimerListener(String context, String url){ this.context = context; This.url = url; } public void timerExpired(Timer timer) { System.out.println("Timer fired. Context ="+ ((StockQuoteTimerListener)timer.getTimerListener()) .getContext()); } public String getContext() { return context; } }

Example: Using connections with asynchronous beans:

An asynchronous bean method can use the connections that its creating Java 2 Platform Enterprise Edition (J2EE) component obtained using java:comp resource references.

For more information on resource references, see the topic References. The following is an example of an asynchronous bean that uses connections correctly:

```
class GoodAsynchBean
{
 DataSource ds;
 public GoodAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource
  // as class instance data.
  InitialContext ic = new InitialContext();
  // it is assumed that the created J2EE component has this
  // resource reference defined in its deployment descriptor.
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
 }
 // When the asynchronous bean method is called, get a connection,
 //  use it, then close it.
 void anEventListener()
 {
  Connection c = null;
  try
  {
   c = ds.getConnection();
   // use the connection now...
  }
  finally
  {
   if(c != null) c.close();
  }
 }
}
```

The following example of an asynchronous bean that uses connections incorrectly:

```
class BadAsynchBean
{
 DataSource ds;
 // Do not do this. You cannot cache connections across asynch method calls.
```

```
 Connection c;

 public BadAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource as
  // class instance data.
  InitialContext ic = new InitialContext();
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
  // here, you broke the rules...
  c = ds.getConnection();
 }
 // Now when the asynch method is called, illegally use the cached connection
 // and you likely see J2C related exceptions at run time.
 // close it.
 void someAsynchMethod()
 {
  // use the connection now...
 }
}
```

Work manager service settings

Use this page to enable or disable the work manager service that manages work manager resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Work Manager Service** .

Startup:

Specifies whether the server attempts to start the work manager service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the work manager service automatically.

Cleared
The server does not try to start the work manager service. If work manager resources are to be used on this server, the system administrator must start the work manager service manually or select this property then restart the server.

Configuring timer managers

A timer manager acts as a thread pool for application components that use asynchronous beans. Use the administrative console to configure timer managers. The timer manager service is enabled by default.

If you are not familiar with timer managers, review the conceptual section, Timer managers, in the Asynchronous beans topic.

You can define multiple timer managers for each cell. Each timer manager is bound to a unique place in Java Naming and Directory Interface (JNDI).

Note: The timer manager service is only supported from within the Enterprise Java Beans (EJB) container or Web container. Looking up and using a configured timer manager from a J2EE application client container is not supported.

1. Start the administrative console.
2. Select **Resources > Asynchronous beans > Timer managers**.

3. Specify a **Scope** value and click **New**.
4. Specify the following required properties:
   Scope The scope of the configured resource. This value indicates the location for the configuration file.
   Name The display name for the timer manager.
   JNDI Name
   The Java Naming and Directory Interface (JNDI) name for the timer manager. This name is used by asynchronous beans that need to look up the timer manager. Each timer manager must have a unique JNDI name within the cell.
   Number of Timer Threads
   The maximum number of threads that are used for timers.
5. [Optional] Specify a **Description** and a **Category** for the timer manager.
6. [Optional] Select the **Service Names** (J2EE contexts) on which you want this timer manager to be made available. Any asynchronous beans that use this timer manager then inherit the selected J2EE contexts from the component that creates the bean. The list of selected services also is known as the ″sticky″ context policy for the timer manager. Selecting more services than are actually required might impede performance.
7. Save your configuration.

The timer manager is now configured and ready for access by application components that need to manage the start of asynchronous code.

Timer manager collection

Use this page to view the configuration properties of timer managers, which enable applications to schedule future timer notifications and to receive timer notification callbacks to application-specified listeners within a Java 2 Platform, Enterprise Edition (J2EE) environment. The timer manager binds to the Java Naming and Directory Interface (JNDI) name space.

A timer manager contains a pool of threads bound into JNDI.

To view this administrative console page, click **Resources** > **Asynchronous beans** < **Timer managers**.

Name:

Specifies the name by which the timer manager is known for administrative purposes.

Data type String

JNDI Name:

Specifies the JNDI name used to look up the timer manager in the name space.

Data type String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Description:

Specifies a description of this timer manager for administrative purposes.

Data type String

Category:

Specifies a string that can be used to classify or group this timer manager.

Data type String

Timer manager settings:

Use this page to modify timer manager settings. Timer managers enable applications to schedule future timer notifications and to receive timer notification callbacks to application-specified listeners within a Java 2 Platform, Enterprise Edition (J2EE) environment. The timer manager binds to the Java Naming and Directory Interface (JNDI) name space.

A timer manager contains a pool of threads bound into JNDI.

To view this administrative console page, click **Resources** > **Asynchronous beans** > **Timer managers** *timermanager_name*.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

Specifies the name by which the timer manager is known for administrative purposes.

Data type String

JNDI Name:

Specifies the JNDI name used to look up the timer manager in the namespace.

Data type String

Description:

Specifies adescription of this timer manager for administrative purposes.

Data type String

Category:

Specifies a string that can be used to classify or group this timer manager.

Data type String

Service Names:

Specifies a list of services to make available to this timer manager.

Asynchronous beans can inherit J2EE context information by enabling one or more J2EE service contexts on the timer manager resource in the WebSphere administrative console or by setting the serviceNames attribute of the TimerManagerInfo configuration object. When specifying the serviceNames attribute each enabled service should be separated by a semicolon. For example:

security;UserWorkArea;com.ibm.ws.i18n. When a J2EE service context is enabled, it propagates the context from the scheduling thread to the target thread. If not enabled, the target thread does not inherit the context of the scheduling thread and a default context is applied. Any related J2EE context that is already present on the thread is suspended before any new J2EE context is applied.

The context information of each selected service is propagated to each timer that is created using this timer manager. Selecting services that are not needed can negatively impact performance.

Work area
Use the administrative console or the UserWorkArea service name to enable work area partitions. When enabled, the work area context for every work area partition that exists on the scheduling thread is available on the target thread. This feature is optional.

Security
Use the administrative console or the security service name to enable the Java Authentication and Authorization Service (JAAS) subject. When this feature and administrative security are enabled, the JAAS subject that is present on the scheduling thread is applied to the target thread. If not enabled, the target thread is run anonymously without a JAAS subject on the thread. This feature is optional.

Internationalization
Use the administrative console or the com.ibm.ws.i18n service name to enable the internationalization context information. When the internationalization context and the Internationalization service is enabled, the internationalization context that exists on the scheduling thread is available on the target thread. This feature is optional.

Number of Timer Threads:

Specifies the maximum number of threads that are used for timers.

Data type Integer

Configuring work managers

A work manager acts as a thread pool for application components that use asynchronous beans. Use the administrative console to configure work managers.

If you are not familiar with work managers, review the conceptual topic, Work managers.

The work manager service is always enabled. In previous versions of the product, the work manager service could be disabled using the administration console or configuration service. The work manager service configuration objects are still present in the configuration service, but the enabled attribute is ignored.

You can define multiple work managers for each cell. Each work manager is bound to a unique place in Java Naming and Directory Interface (JNDI).

Note: The work manager service is only supported from within the Enterprise Java Beans (EJB) Container or Web Container. Looking up and using a configured work manager from a J2EE application client container is not supported.

1. Start the administrative console.
2. Select **Resources > Asynchronous beans > Work managers**.
3. Specify a **Scope** value and click **New**.

4. Specify the required properties for work manager settings.
   Scope The scope of the configured resource. This value indicates the location for the configuration file.
   Name The display name for the work manager.
   JNDI Name
   The Java Naming and Directory Interface (JNDI) name for the work manager. This name is used by asynchronous beans that need to look up the work manager. Each work manager must have a unique JNDI name within the cell.
   Number of Alarm Threads
   The maximum number of threads to use for processing alarms. A single thread is used to monitor pending alarms and dispatch them. An additional pool of threads is used for dispatching the threads. All alarm managers on the asynchronous beans associated with this work manager share this set of threads. A single alarm thread pool exists for each work manager, and all of the asynchronous beans associated with the work manager share this pool of threads.
   Minimum Number Of Threads
   The initial number of threads to be created in the thread pool.
   Maximum Number Of Threads
   The maximum number of threads to be created in the thread pool. The maximum number of threads can be exceeded temporarily if the **Growable** check box is selected. These additional threads are discarded when the work on the thread completes.
   Thread Priority
   The priority to assign to all threads in the thread pool.

   Every thread has a priority. Threads with higher priority are run before threads with lower priority. For more information about how thread priorities are used, see the javadoc for the setPriority method of the java.lang.Thread class in the Java Standard Edition specification.
5. [Optional] Specify a **Description** and a **Category** for the work manager.
6. [Optional] Select the **Service Names** (J2EE contexts) on which you want this work manager to be made available. Any asynchronous beans that use this work manager then inherit the selected J2EE contexts from the component that creates the bean. The list of selected services also is known as the ″sticky″ context policy for the work manager. Selecting more services than are actually required might impede performance.

   Other optional fields include:

   Work timeout
   Specifies the number of milliseconds to wait before a scheduled work object is released. If a value is not specified, then the timeout is disabled.

   Work request queue size
   Specifies the size of the work request queue. The work request queue is a buffer that holds scheduled work objects and may be any value 1 or greater. The thread pool pulls work from this queue. If you do not specify a value or the value is 0, the queue size is managed automatically. Large values can consume significant system resources.

   Work request queue full action
   Specifies the action taken when the thread pool is exhausted, and the work request queue is full. This action starts when you submit non-daemon work to the work manager. If set to FAIL, the work manager API methods creates an exception instead of blocking.
7. Save your configuration.

The work manager is now configured and ready for access by application components that need to manage the start of asynchronous code.

Work manager collection

Use this page to view the collection properties of work managers, which contain a pool of threads bound into the Java Naming and Directory Interface.

To view this administrative console page, click **Resources > Asynchronous beans > Work managers**.

Name:

Specifies the name by which the work manager is known for administrative purposes.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name used to look up the work manager in the namespace.

Data type String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Description:

Specifies the description of this work manager for administrative purposes.

Category:

Specifies a category name that is used to classify or group this work manager.

Work manager settings:

Use this page to modify work manager settings. Work managers contain a pool of threads bound into Java Naming and Directory Interface.

To view this administrative console page, click **Resources > Asynchronous beans > Work managers >** *workmanager_name*.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

Specifies the name by which the work manager is known for administrative purposes.

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name used to look up the work manager in the namespace.

Description:

Specifies the description of this work manager for administrative purposes.

Category:

Specifies a string that you can use to classify or group this work manager.

Work timeout:

Specifies the number of milliseconds to wait before a scheduled work object is released. If not specified, the timeout is disabled.

Default 0

Work request queue size:

Specifies the size of the work request queue. The work request queue is a buffer that holds scheduled work objects and may be any value 1 or greater. The thread pool pulls work from this queue. If you do not specify a value or the value is 0, the queue size is managed automatically. Large values can consume significant system resources.

Default 0

Work request queue full action:

Specifies the action taken when the thread pool is exhausted, and the work request queue is full. This action starts when you submit non-daemon work to the work manager.

If set to FAIL, the work manager API methods creates an exception instead of blocking.

Default BLOCK
Range FAIL

Service names:

Specifies a list of services to make available to this work manager.

Asynchronous beans can inherit J2EE context information by enabling one or more J2EE service contexts on the work manager resource in the WebSphere administrative console or by setting the serviceNames attribute of the WorkManagerInfo configuration object. When specifying the serviceNames attribute each enabled service should be separated by a semicolon. For example: `security;UserWorkArea;com.ibm.ws.i18n`. When a J2EE service context is enabled, it propagates the context from the scheduling thread to the target thread. If not enabled, the target thread does not inherit the context of the scheduling thread and a default context is applied. Any related J2EE context that is already present on the thread is suspended before any new J2EE context is applied.

The context information of each selected service is propagated to each work or alarm that is created using this work manager. Selecting services that are not needed can negatively impact performance.

Application profile (deprecated)

Use the administrative console or the AppProfileService service name to enable the application profile tasks. Application profile context is not supported and not available for J2EE 1.4 applications. For J2EE 1.3 applications, the application profile context is deprecated and is only available when **Application Profile Service 5.x Compatibility Mode** is enabled and both the scheduling thread and target thread are J2EE 1.3 applications. When enabled, all application profile tasks that are available on the scheduling thread are available on the target thread. The scheduled work that runs in a J2EE 1.4 application does not get the application profiling task of the scheduling thread. This feature is optional.

Work area	Use the administrative console or the `UserWorkArea` service name to enable work area partitions. When enabled, the work area context for every work area partition that exists on the scheduling thread is available on the target thread. This feature is optional.
Security	Use the administrative console or the `security` service name to enable the Java Authentication and Authorization Service (JAAS) subject. When this feature and administrative security are enabled, the JAAS subject that is present on the scheduling thread is applied to the target thread. If not enabled, the target thread is run anonymously without a JAAS subject on the thread. This feature is optional.
Internationalization	Use the administrative console or the `com.ibm.ws.i18n` service name to enable the internationalization context information. When the internationalization context and the Internationalization service is enabled, the internationalization context that exists on the scheduling thread is available on the target thread. This feature is optional.

Thread pool properties:

Specifies the priority of the threads available in this work manager.

Number of alarm threads	Specifies the desired maximum number of threads used for alarms. The default value is 2.
Minimum number of threads	Specifies the minimum number of threads available in this work manager.
Maximum number of threads	Specifies the maximum number of threads available in this work manager.
Thread priority	Specifies the priority of the threads available in this work manager.
Growable	Specifies whether the number of threads in this work manager can be increased.

Dynamic cache

Task overview: Using the dynamic cache service to improve performance

Use the dynamic cache service to improve application performance by caching the output of servlets, commands, and JavaServer Pages (JSP) files.

The dynamic cache service works within an application server Java virtual machine (JVM), intercepting calls to cacheable objects. For example, it intercepts calls through a servlet service method or a command execute method, and either stores the output of the object to the cache or serves the content of the object from the dynamic cache.

1. Enable the dynamic cache service globally. To use the features associated with dynamic caching, you must enable the service in the administrative console. See “Enabling the dynamic cache service” on page 1929 for more information.
2. Configure the type of caching that you are using:
   - “Configuring servlet caching” on page 1933.
   - “Configuring portlet fragment caching” on page 1933.

- "Configuring Edge Side Include caching" on page 1940.
- "Configuring command caching" on page 1960.
- "Example: Caching Web services" on page 1915.
- "Configuring the Web services client cache" on page 1962.

3. Monitor the results of your configuration using the dynamic cache monitor. For more information, see "Displaying cache information" on page 1977.
4. If you have any problems with your configuration, see the *Troubleshooting and support* PDF.

To use the DistributedMap and DistributedObjectCache interfaces for the dynamic cache, see "Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache" on page 1968.

Dynamic cache

Caching the output of servlets, commands, and JavaServer Pages (JSP) improves application performance. WebSphere Application Server consolidates several caching activities including servlets, Web services, and WebSphere commands into one service called the *dynamic cache*. These caching activities work together to improve application performance, and share many configuration parameters that are set in the dynamic cache service of an application server.

You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, the results of a servlet after it runs, and metadata.

Eviction policies using the disk cache garbage collector

The disk cache garbage collector is responsible for evicting objects out of the disk cache, based on a specified eviction policy.

The garbage collector keeps a certain amount of space on disk available, which is governed by the configuration attribute that limits the amount of disk space that is used for caching objects. To enable the eviction policy, enable the Limit disk cache size in GB and/or Limit disk cache size in entries options in the administrative console.

The garbage collector is triggered when the disk space reaches a specified high threshold (a percentage of the Limit disk cache size in entries or in GB) and evicts objects, based on the eviction policy, from the disk in the background until the disk cache size reaches a specified low threshold (a percentage of the Limit disk cache size in entries or in GB). Eviction triggers when one or both of the high thresholds is reached for Limit disk cache size in GB and Limit disk cache size in entries. The supported policies are:

- None: This is the default policy. Objects are evicted only when they expire, or if they are invalidated.
- Random: The expired objects are removed first. If the disk size still has not reached the low threshold limit, objects are picked from the disk cache in random order and removed until the disk size reaches a low threshold limit.
- Size: The expired objects are removed first. If the disk size still has not reached the low threshold limit, then largest-sized objects are removed until the disk size reaches a low threshold limit.

Limit disk cache size in GB and High Threshold determines when to trigger eviction and when the disk cache is considered near full. It is computed as a function of the user-specified limit. If the specified limit is *10* GB (3 GB is the minimum), the cache subsystem initially creates three files that can grow to 1 GB in size for cache data, dependency ID information, and template information. Each time more space is needed to contain cache data, dependency ID information, or template information, a new file is created. Each of these files grow in 1 GB increments until the total number of files that are created is equal to disk cache in size in GB (in this case ten). Although the initial size of the new file may be much smaller than 1 GB, the dynamic cache service always rounds up to the next GB.

Eviction triggers when the cache data size reaches the high threshold and continues until the cache data size reaches the low threshold. Calculation of cache data size is dynamic. The following formula describes how to calculate the actual cache data size limit:

```
cache data size limit = disk cache size (in GB) - number of dependency files per GB - number of template files
```

When the cache data size limit is defined, the trigger point is calculated as follows:

```
eviction trigger point = cache data size limit * high threshold
size of evicted entries = cache data size * (high threshold - low threshold)
```

Consider the following scenarios:

- **Scenario 1**
  - Disk cache size in GB = 10 GB
  - High threshold = 90%
  - Low Threshold = 80%

  Initially, there is one file for dependency ID and template ID.

  ```
  cache data size limit = 10-(1+1) = 8 GB
  eviction trigger point = 8 * 90% = 7.2 GB
  size of evicted entries = 8 * (90% - 80%) = 0.8 GB
  ```

  In the above scenario, eviction starts when the data cache size reaches 7.2 GB and continues until the cache size is 6.4 GB (7.2 - 0.8).
- **Scenario 2**

  In scenario 1, if the dependency files grow to more than 1 GB, an additional dependency file generates. The eviction trigger point launches dynamically as follows:

  ```
  cache data size limit = 10 - (2+1) = 7GB
  eviction trigger point = 7 * 90% = 6.3GB
  size of evicted entries = 7 * (90% - 80%) = 0.7GB
  ```

  In the above scenario, eviction starts when the data cache size reaches 6.3 GB, and continues until the cache size in 5.6 GB (6.3 - 0.7).

Disk cache eviction for limit disk cache size in entries. Consider the following scenario:

- Disk cache size in entries = 100000
- High threshold = 90%
- Low threshold = 80%

```
eviction trigger point = 100000 * 90% = 90000
number of entries evicted = 100000 * (90% - 80%) = 10000
```

In this scenario, eviction starts when the number of cache entries reaches 90000 and 10000 entries are evicted from the cache.

Example: Caching Web services

This topic includes examples of building a set of cache policies and SOAP messages for a Web services application.

The following is a example of building a set of cache policies for a simple Web services application. The application in this example stores stock quotes and has operations to read, update the price of, and buy a given stock symbol.

Following are two SOAP message examples that the application can receive, with accompanying HTTP Request headers.

The first message sample contains a SOAP message for a GetQuote operation, requesting a quote for IBM. This is a read-only operation that gets its data from the back end, and is a good candidate for caching. In this example the SOAP message is cached and a timeout is placed on its entries to guarantee the quotes it returns are current.

Message example 1

```
POST /soap/servlet/soaprouter
HTTP/1.1
Host: www.myhost.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: urn:stockquote-lookup
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:getQuote xmlns:m="urn:stockquote">
<symbol>IBM</symbol>
</m:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

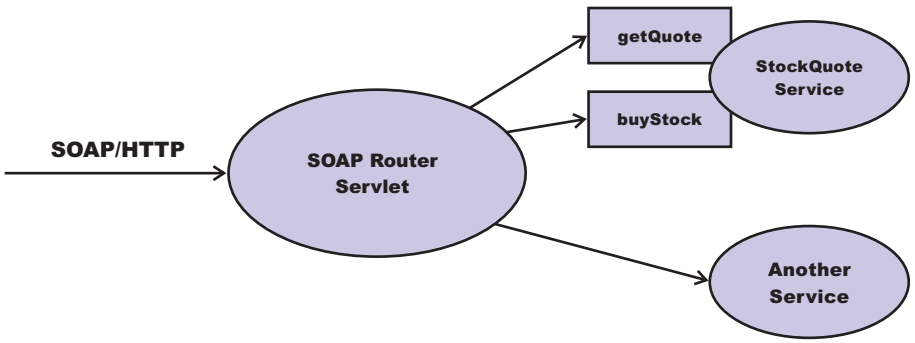
The SOAPAction HTTP header in the request is defined in the SOAP specification and is used by HTTP proxy servers to dispatch requests to particular HTTP servers. WebSphere Application Server dynamic cache can use this header in its cache policies to build IDs without having to parse the SOAP message.

Message example 2 illustrates a SOAP message for a BuyQuote operation. While message 1 is cacheable, this message is not, because it updates the back end database.

Message example 2

```
POST /soap/servlet/soaprouter
HTTP/1.1
Host: www.myhost.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: urn:stockquote-update
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:buyStock xmlns:m="urn:stockquote">
<symbol>IBM</symbol>
</m:buyStock>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following graphic illustrates how to invoke methods with the SOAP messages. In Web services terms, especially Web Service Definition Language (WSDL), a service is a collection of operations such as getQuote and buyStock. A body element namespace (urn:stockquote in the example) defines a service, and the name of the first body element indicates the operation.



The following is an example of WSDL for the getQuote operation:

```
<?xml version="1.0"?>
<definitions name="StockQuoteService-interface"
targetNamespace="http://www.getquote.com/StockQuoteService-interface"
xmlns:tns="http://www.getquote.com/StockQuoteService-interface"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns=soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
<message name="SymbolRequest">
<part name="return" type="xsd:string"/>
</message>
<portType name="StockQuoteService">
<operation name="getQuote">
<input message="tns:SymbolRequest"/>
<output message="tns:QuoteResponse"/>
</operation>
</portType>
<binding name="StockQuoteServiceBinding"
type="tns:StockQuoteService">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getQuote">
<soap:operation soapAction="urn:stockquote-lookup"/>
<input>
<soap:body use="encoded" namespace="urn:stockquote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded" namespace="urn:stockquotes"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>
</definition>
```

To build a set of cache policies for a Web services application, configure WebSphere Application Server dynamic cache to recognize cacheable service operation of the operation.

WebSphere Application Server inspects the HTTP request to determine whether or not an incoming message can be cached based on the cache policies defined for an application. In this example, buyStock and stock-update are not cached, but stockquote-lookup is cached. In the `cachespec.xml` file for this Web application, the cache policies need defining for these services so that the dynamic cache can handle both SOAPAction and service operation.

WebSphere Application Server uses the operation and the message body in Web services cache IDs, each of which has a component associated with them. Therefore, each Web services `<cache-id>` rule contains only two components. The first is for the operation. Because you can perform the stockquote-lookup operation by either using a SOAPAction header or a service operation in the body, you must define two different `<cache-id>` elements, one for each method. The second component is of type ″body″, and defines how WebSphere Application Server should incorporate the message body into the cache ID. You can use a hash of the body, although it is legal to use the literal incoming message in the ID.

The incoming HTTP request is analyzed by WebSphere Application Server to determine which of the `<cache-id>` rules match. Then, the rules are applied to form cache or invalidation IDs.

The following is sample code of a `cachespec.xml` file defining SOAPAction and servicesOperation rules:

```
<cache>
<cache-entry>
 <class>webservice</class>
 <name>/soap/servlet/soaprouter</name>
 <sharing-policy>not-shared</sharing-policy>
 <cache-id>
```

```
  <component id="" type="SOAPAction">
   <value>urn:stockquote-lookup</value>
  </component>
  <component id="Hash" type="SOAPEnvelope"/>
   <timeout>3600</timeout>
   <priority>1<priority>
  </component>
 </cache-id>
 <cache-id>
  <component id="" type="serviceOperation">
   <value>urn:stockquote:getQuote</value>
  </component>
  <component id="Hash" type="SOAPEnvelope"/>
   <timeout>3600</timeout>
   <priority>1</priority>
  </component>
 </cache-id>
</cache-entry>
</cache>
```

Dynamic cache MBean statistics

The dynamic cache service provides an MBean interface to access cache statistics.

Access cache statistics with the MBean interface, using JACL

- Obtain the MBean identifier with the **queryNames** command, for example:

  ```
  $AdminControl queryNames type=DynaCache,*  // Returns a list of the available dynamic cache MBeans
  ```

  Select your dynamic cache MBean and run the following command:

  ```
  set mbean <dynamic_cache_mbean>
  ```

- Retrieve the names of the available cache statistics:

  ```
  $AdminControl invoke $mbean getCacheStatisticNames
  ```

- Retrieve the names of the available cache instances:

  ```
  $AdminControl invoke $mbean getCacheInstanceNames
  ```

- Retrieve all of the available cache statistics for the base cache instance:

  ```
  $AdminControl invoke $mbean getAllCacheStatistics
  ```

- Retrieve all of the available cache statistics for the named cache instance:

  ```
  $AdminControl invoke $mbean getAllCacheStatistics "services/cache/servletInstance_4"
  ```

- Retrieve cache statistics that are specified by the names array for the base cache instance:

  ```
  $AdminControl invoke $mbean getCacheStatistics
   {"DiskCacheSizeInMB ObjectsReadFromDisk4000K RemoteObjectMisses"}
  ```

  Note: This command should all be entered on one line. It is broken here for printing purposes.

- Retrieve cache statistics that are specified by the names array for the named cache instance:

  ```
  $AdminControl invoke $mbean getCacheStatistics
   {services/cache/servletInstance_4 "ExplicitInvalidationsLocal CacheHits"}
  ```

  Note: This command should all be entered on one line. It is broken here for printing purposes.

Example: Configuring the dynamic cache

This example puts all the steps together for configuring the dynamic cache with the `cachespec.xml` file, showing the use of the cache ID generation rules, dependency IDs, and invalidation rules.

Suppose that a servlet is used to manage a simple news site. This servlet uses the query parameter ″action″ to determine if the request is being used to ″view″ news or ″update″ news (used by the administrator). Another query parameter ″category″ is used to select the news category. Suppose that this site supports an optional customized layout that is stored in the user's session using the attribute name ″layout″. Here are example URL requests to this servlet:

http://*yourhost/yourwebapp*/newscontroller?action=view&category=sports (Returns a news page for the sports category)

http://*yourhost/yourwebapp*/newscontroller?action=view&category=money (Returns a news page for the money category)

http://*yourhost/yourwebapp*/newscontroller?action=update&category=fashion (Allows the administrator to update news in the fashion category)

Here are the steps for configuring dynamic cache for this example with the `cachespec.xml` file:

1. Define the <cache-entry> elements necessary to identify the servlet. In this case, the servlet's URI is ″newscontroller″ so this is the cache-entry's <name> element. Because this example caches a servlet or JavaServer Pages (JSP) file, the cache entry class is ″servlet″.

```
<cache-entry>
<name> /newscontroller </name>
<class>servlet  </class>
 </cache-entry>
```

2. Define cache ID generation rules. This servlet is cached only when action=view, so one component of the cache ID is the parameter ″action″ when the value equals ″view″. The news category is also an essential part of the cache ID. Finally, the optional session attribute for the user's layout is included in the cache ID. The cache entry now is :

```
<cache-entry>
 <name> /newscontroller </name>
 <class>servlet  </class>
  <cache-id>
  <component id="action" type="parameter">
   <value>view</value>
   <required>true</required>
  </component>
  <component id="category" type="parameter">
   <required>true</required>
  </component>
  <component id="layout" type="session">
   <required>false</required>
  </component>
 </cache-id>
</cache-entry>
```

3. Define dependency ID rules. For this servlet, a dependency ID is added for the category. Later, when the category is invalidated due to an update event, all views of that news category are invalidated. Following is an example of the cache entry after adding the dependency-id:

```
<cache-entry>
 <name>newscontroller </name>
 <class>servlet  </class>
  <cache-id>
  <component id="action" type="parameter">
   <value>view</value>
   <required>true</required>
  </component>
  <component id="category" type="parameter">
   <required>true</required>
  </component>
  <component id="layout" type="session">
   <required>false</required>
  </component>
 </cache-id>
 <dependency-id>category
  <component id="category" type="parameter">
   <required>true</required>
  </component>
 </dependency-id>
</cache-entry>
```

4. Define invalidation rules. Since a category dependency ID is already defined, define an invalidation rule to invalidate the category when action=update. To incorporate the conditional logic, we will add ″ignore-value″ components into the invalidation rule. These components do not add to the output of the invalidation ID, but only determine whether or not the invalidation ID is created and run. The final cache-entry now looks like this:

```
<cache-entry>
 <name>newscontroller </name>
 <class>servlet  </class>
  <cache-id>
  <component id="action" type="parameter">
   <value>view</value>
   <required>true</required>
  </component>
  <component id="category" type="parameter">
   <required>true</required>
  </component>
  <component id="layout" type="session">
   <required>false</required>
  </component>
 </cache-id>
 <dependency-id>category
  <component id="category" type="parameter">
   <required>true</required>
  </component>
 </dependency-id>
 <invalidation>category
  <component id="action" type="parameter" ignore-value="true">
   <value>update</value>
   <required>true</required>
  </component>
  <component id="category" type="parameter">
   <required>true</required>
     </component>
 </invalidation>
</cache-entry>
```

Accessing dynamic cache PMI counters

The dynamic cache statistics interface is defined as WSDynamicCacheStats under the `com\ibm\websphere\pmi\stat` package.

Dynamic cache statistics are structured as follows in the Performance Monitoring Infrastructure (PMI) tree:

```
__Dynamic Caching+
  |
  |__<Servlet: instance_1>
    |__Templates+
      |__<template_1>
      |__<template_2>
    |__Disk+
      |__<Disk Offload Enabled>
  |
  |__<Object: instance_2>
    |__Object Cache+
      |__<Counters>
+ indicates logical group
```

StatDescriptor locates and accesses particular statistics in the PMI tree. For example:

1. StatDescriptor to represent statistics for cache servlet: `instance_1 templates group template_1: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, ″Servlet: instance1″, WSDynamicCacheStats.TEMPLATE_GROUP, ″template_1″});`
2. StatDescriptor to represent statistics for cache servlet: `instance_1 disk group Disk Offload Enabled: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, ″Servlet: instance_1″, WSDynamicCacheStats.DISK_GROUP, WSDynamicCacheStats.DISK_OFFLOAD_ENABLED});`

3. StatDescriptor to represent statistics for cache object: `instance2 object cache group Counters: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, "Object: instance_2", WSDynamicCacheStats.OBJECT_GROUP, WSDynamicCacheStats.OBJECT_COUNTERS});`

Important: Cache instance names are prepended with cache type ("Servlet: " or "Object: ").

Counter definitions for Servlet Cache

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. MaxInMemoryCache EntryCount	`WSDynamicCacheStats.NAME - "Servlet: instance_1"`	The maximum number of in-memory cache entries.	5.0 and later
WSDynamicCacheStats. InMemoryCache EntryCount	`WSDynamicCacheStats.NAME - "Servlet: instance_1"`	The current number of in-memory cache entries	5.0 and later
WSDynamicCacheStats. HitsIn MemoryCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of requests for cacheable objects that are served from memory.	5.0 and later
WSDynamicCacheStats. HitsOnDiskCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of requests for cacheable objects that are served from disk.	5.0 and later
WSDynamicCacheStats. ExplicitInvalidationCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of explicit invalidations.	5.0 and later
WSDynamicCacheStats. LruInvalidationCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1`	The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. instance.	5.0 and later
WSDynamicCacheStats. TimeoutInvalidationCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of cache entries that are removed from memory and disk because their timeout has expired.	5.0 and later
WSDynamicCacheStats. InMemoryAndDisk CacheEntryCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The current number of used cache entries in memory and disk.	5.0 and later
WSDynamicCacheStats. RemoteHitCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain.	5.0 and later

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. MissCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of requests for cacheable objects that were not found in the cache.	5.0 and later
WSDynamicCacheStats. ClientRequestCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of requests for cacheable objects that are generated by applications running on this application server.	5.0 and later
WSDynamicCacheStats. DistributedRequestCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of requests for cacheable objects that are generated by cooperating caches in this replication domain.	5.0 and later
WSDynamicCacheStats. ExplicitMemory InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations resulting in the removal of an entry from memory.	5.0 and later
WSDynamicCacheStats. ExplicitDisk InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations resulting in the removal of an entry from disk.	5.0 and later
WSDynamicCacheStats. LocalExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations generated locally, either programmatically or by a cache policy.	5.0 and later
WSDynamicCacheStats. RemoteExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations received from a cooperating Java virtual machine in this replication domain.	5.0 and later
WSDynamicCacheStats. RemoteCreationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of cache entries that are received from cooperating dynamic caches.	5.0 and later

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ObjectsOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of cache entries on disk.	6.1
WSDynamicCacheStats. HitsOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of requests for cacheable objects that are served from disk.	6.1
WSDynamicCacheStats. ExplicitInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of explicit invalidations resulting in the removal of entries from disk.	6.1
WSDynamicCacheStats. TimeoutInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of disk timeouts.	6.1
WSDynamicCacheStats PendingRemoval FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of pending entries that are to be removed from disk.	6.1
WSDynamicCacheStats. DependencyIdsOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency ID that are on disk.	6.1
WSDynamicCacheStats. DependencyIdsBuffered ForDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency IDs that are buffered for the disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. DependencyIds OffloadedToDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency IDs that are offloaded to disk.	6.1
WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency ID-based invalidations.	6.1
WSDynamicCacheStats. TemplatesOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are on disk.	6.1
WSDynamicCacheStats. TemplatesBuffered ForDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are buffered for the disk.	6.1
WSDynamicCacheStats. TemplatesOffloaded ToDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of templates that are offloaded to disk.	6.1
WSDynamicCacheStats. TemplateBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of template-based invalidations.	6.1
WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached.	6.1
WSDynamicCacheStats. OverflowInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1 " - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit.	6.1

Counter definitions for Object Cache

Name of PMI Statistics	Path	Description	Version
WSDynamicCacheStats. MaxInMemoryCache EntryCount	WSDynamicCacheStats.NAME - "Object: instance_2"	The maximum number of in-memory cache entries.	5.0 and later
WSDynamicCacheStats. InMemoryCache EntryCount	WSDynamicCacheStats.NAME - "Object: instance_2"	The current number of in-memory cache entries	5.0 and later
WSDynamicCacheStats. HitsInMemoryCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats.OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are served from memory.	5.0 and later
WSDynamicCacheStats. HitsOnDiskCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are served from disk.	5.0 and later
WSDynamicCacheStats. ExplicitInvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations.	5.0 and later
WSDynamicCacheStats. LruInvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. instance.	5.0 and later
WSDynamicCacheStats. TimeoutInvalidation Count	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of cache entries that are removed from memory and disk because their timeout has expired.	5.0 and later
WSDynamicCacheStats. InMemoryAndDisk CacheEntryCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The current number of used cache entries in memory and disk.	5.0 and later

Name of PMI Statistics	Path	Description	Version
WSDynamicCacheStats. RemoteHitCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain.	5.0 and later
WSDynamicCacheStats. MissCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that were not found in the cache.	5.0 and later
WSDynamicCacheStats. ClientRequestCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are generated by applications running on this application server.	5.0 and later
WSDynamicCacheStats. DistributedRequest Count	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are generated by cooperating caches in this replication domain.	5.0 and later
WSDynamicCacheStats. ExplicitMemory InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations resulting in the removal of an entry from memory.	5.0 and later
WSDynamicCacheStats. ExplicitDisk InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations resulting in the removal of an entry from disk.	5.0 and later
WSDynamicCacheStats. LocalExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations generated locally, either programmatically or by a cache policy.	5.0 and later
WSDynamicCacheStats. RemoteExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations received from a cooperating Java virtual machine in this replication domain.	5.0 and later

Name of PMI Statistics	Path	Description	Version
WSDynamicCacheStats. RemoteCreationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of cache entries that are received from cooperating dynamic caches.	5.0 and later

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ObjectsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of cache entries on disk.	6.1
WSDynamicCacheStats. HitsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of requests for cacheable objects that are served from disk.	6.1
WSDynamicCacheStats. ExplicitInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of explicit invalidations resulting in the removal of entries from disk.	6.1
WSDynamicCacheStats. TimeoutInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of disk timeouts.	6.1
WSDynamicCacheStats PendingRemoval FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of pending entries that are to be removed from disk.	6.1
WSDynamicCacheStats. DependencyIdsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency ID that are on disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. DependencyIds BufferedForDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency IDs that are buffered for the disk.	6.1
WSDynamicCacheStats. DependencyIds OffloadedToDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency IDs that are offloaded to disk.	6.1
WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats.DISK_ OFFLOAD_ENABLED	The number of dependency ID-based invalidations.	6.1
WSDynamicCacheStats. TemplatesOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are on disk.	6.1
WSDynamicCacheStats. TemplatesBuffered ForDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP / -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are buffered for the disk.	6.1
WSDynamicCacheStats. TemplatesOffloaded ToDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of templates that are offloaded to disk.	6.1
WSDynamicCacheStats. TemplateBasedInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of template-based invalidations.	6.1
WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. OverflowInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit.	6.1

Enabling the dynamic cache service

Enable the dynamic cache service to improve application performance by caching the output of servlets, Web services, and WebSphere Application Server commands into memory.

Develop a cache policy for your application. The cache policy defines rules for what responses to cache and the amount of time the responses should be held in the cache. See “Configuring cacheable objects with the cachespec.xml file” on page 1946 for more information.

The dynamic cache service is enabled by default. However, you can enable or disable the service through the administrative console.

1. Open the administrative console.
2. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Container services > Dynamic cache service**.
3. Select **Enable service at server startup** .
4. Click **Apply** or **OK**.
5. Restart WebSphere Application Server. You might want to enable servlet caching before restarting WebSphere Application Server. See “Configuring servlet caching” on page 1933 for more information.

The dynamic cache service caches content for requests that have cache policies configured.

You might want to enable dynamic cache disk offload. This option moves cache entries that are expired from memory to disk for potential future access. See “Configuring dynamic cache disk offload” on page 1937 for more information.

Dynamic cache service settings

Use this page to configure and manage the dynamic cache service settings.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Container services > Dynamic cache service**.

Enable service at server startup:

Specifies whether the dynamic cache is enabled when the server starts.

Cache size:

Specifies a positive integer as the value for the maximum number of entries the cache holds.

Enter the cache size value in this field between the range of 100 through 200,000.

Default priority:

Specifies the default priority for cache entries, determining how long an entry stays in a full cache.

Default	1
Range	1 to 255

Enable disk offload:

Specifies whether disk offload is enabled.

By default, the dynamic cache maintains the number of entries configured in memory. If new entries are created while the cache is full, the priorities that are configured for each cache entry and a least recently used algorithm are used to remove entries from the cache. In addition to having a cache entry removed from memory when the cache is full, you can enable disk offload to have a cache entry copied to the file system (the location is configurable). Later, if that cache entry is needed, it is moved back to memory from the file system.

Before you enable disk offload, consider the following:

- You cannot specify the number of cache entries that are offloaded to disk.
- You cannot specify the amount of disk space to use.

Offload location:

Specifies the location on the disk to save cache entries when disk offload is enabled.

If disk offload location is not specified, the default location, `${WAS_TEMP_DIR}/node/server name/_dynacache/cache JNDI name` will be used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, `${USER_INSTALL_ROOT}/diskoffload` generates the location as `${USER_INSTALL_ROOT}/diskoffload/node/server name/cache JNDI name`. This value is ignored if disk offload is not enabled.

The default value of the ${WAS_TEMP_DIR} property is `${USER_INSTALL_ROOT}/temp`. If you change the value of the ${WAS_TEMP_DIR} property after starting WebSphere Application Server, but do not move the disk cache contents to the new location:

- The application server creates a new disk cache file at the new disk offload location.
- If the Flush to disk setting is enabled, all the disk cache content at the old location is lost when you restart the Application Server

When you are specifying a directory, consider the following:

- If you expect to cache a large number of objects or large objects that will be around for some time, consider using a separate disk drive if you are using Windows operating systems, or a separate file system if you are using UNIX platforms.
- If you use the default directory and the disk fills up, WebSphere Application Server could possibly stall if it needs to write messages to log files, and there is no more space.
- If you specify a directory such as `/tmp` on UNIX platforms and that directory fills up, you may have trouble logging onto the system.
- Depending on the operating system, you may see disk full messages on the console.

Flush to disk:

Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if **Enable disk offload** is not selected.

Default	false

Limit disk cache size in GB:

Specifies a value for the maximum disk cache size in GB. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	3 and above.

Limit disk cache size in entries:

Specifies a value for the maximum disk cache size in number of entries. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache entry size:

Specifies a value for the maximum size of an individual cache entry in MB. Any cache entry larger than this, when evicted from memory, will not be offloaded to disk. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Disk cache performance settings:

Specifies the level of performance that is required by the disk cache. This setting applies only if **enableDiskOffload** is specified for the cache. Performance levels determine how memory resources should be used on background activity such as cache cleanup, expiration, garbage collection, and so on. This setting applies only if enable disk offload is specified for the cache.

High performance and high memory usage	Indicates that all metadata will be kept in memory.
Balanced performance and balanced memory usage	Indicates some metadata will be kept in memory. This is the default performance setting and will provide an optimal balance of performance and memory usage for most users.
Low performance and low memory usage	Indicates that limited metadata will be kept in memory.
Custom performance	Indicates that the administrator will explicitly configure the memory settings that will be used to support the above background activity. The administrator sets these values using the **DiskCacheCustomPerformanceSettings** object.

Disk cache cleanup frequency:

Specifies a value for the disk cache cleanup frequency, in minutes. If this value is set to 0, the cleanup runs only at midnight. This setting applies only when the Disk Offload Performance Level is low, balanced, or custom. The high performance level does not require disk cleanup, and this value is ignored.

Value	0 to 1440

Maximum buffer for cache identifiers per metaentry:

Specifies a value for the maximum number of cache identifiers that are stored for an individual dependency ID or template in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk offload performance level is CUSTOM.

Value	100 to MAXINT

Maximum buffer for dependency identifiers:

Specifies a value for the maximum number of dependency identifier buckets in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	100 to MAXINT

Maximum buffer for templates:

Specifies a value for the maximum number of template buckets that are in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	10 to MAXINT

Disk cache eviction algorithm:

Specifies the eviction algorithm that the disk cache will use to evict entries once the high threshold is reached. This setting applies only if enable disk offload is specified for the cache. This setting does not apply when the disk cache eviction policy is set to none.

None	No eviction policy, so the disk cache can grow until it reaches its limit at which time the dynamic cache service stops writing to disk
Random	When the disk size reaches a high threshold limit, the disk cache garbage collector wakes up and randomly picks entries on the disk and evicts them until the size reaches a low threshold limit.
Size	When the disk size reaches a high threshold limit, the disk cache garbge collector wakes up and picks the largest entries on the disk and evicts them until the disk size reaches a low threshold limit.

High threshold:

Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value is used when limit disk cache size in GB and limit disk cache size in entries are specified. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Low threshold:

Specifies when the eviction policy will end. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value is used limit disk cache size in GB and limit disk cache size in entries are specified. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Configuring servlet caching

Configure servlet caching to save the output of servlets and JavaServer Pages (JSP) files to the dynamic cache.

To enable servlet caching, you must complete “Enabling the dynamic cache service” on page 1929.

1. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Web container settings > Web container** in the console navigation tree.
2. Select **Enable servlet caching** under the Configuration tab.
3. Click **Apply** or **OK**.
4. Restart WebSphere Application Server. See Managing application servers for more information.

Define the cache policy for your servlets by “Configuring cacheable objects with the cachespec.xml file” on page 1946.

Servlet caching:

After a servlet is invoked and completes generating the output to cache, a cache entry is created containing the output and the side effects of the servlet. These side effects can include calls to other servlets or JavaServer Pages (JSP) files or metadata about the entry, including timeout and entry priority information.

Unique entries are distinguished by an ID string that is generated from the HttpServletRequest object each time the servlet runs. You can then base servlet caching on:

- Request parameters and attributes of the Universal Resource Identifier (URI) that was used to invoke the servlet
- Session information
- Other options, including cookies

Because JavaServer Pages files are compiled into servlets, the dynamic cache function treats JavaServer Pages files the same as servlets, except in specifically documented situations.

To enable servlet caching see “Configuring servlet caching.” To configure cache policies for your servlets, see “Configuring cacheable objects with the cachespec.xml file” on page 1946.

Configuring portlet fragment caching

Configure portlet fragment caching with the WebSphere Application Sever administrative console to save the output of portlets to the dynamic cache.

To enable portlet fragment caching, you must complete “Enabling the dynamic cache service” on page 1929.

1. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Portlet container settings > Portlet container** in the administrative console navigation tree.
2. Select **Enable portlet fragment cache** under the Configuration tab.
3. Click **Apply** or **OK**.
4. Restart WebSphere Application Server.

   See the *Setting up the application serving environment* PDF for more information.

Define the cache policy for your portlets by “Configuring cacheable objects with the cachespec.xml file” on page 1946.

Portlet fragment caching:

After a portlet is invoked and completes generating the output to cache, a cache entry is created, containing the output and the side effects of the portlet. These side effects can include calls to other portlets or metadata about the entry, including timeout and entry priority information.

Unique entries are distinguished by an ID string that generated from the PortletRequest object each time the portlet runs. You can then base portlet fragment caching on:
- Request parameters and attributes
- Session information
- Portlet-specific information, portlet session, portlet window ID, portlet mode, and portlet window state

To enable portlet fragment caching see “Configuring portlet fragment caching” on page 1933. To configure cache policies for your portlets, see “Configuring cacheable objects with the cachespec.xml file” on page 1946.

Configuring portlet fragment caching with the wsadmin tool

You can configure portlet fragment caching with scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the *Using the administrative clients* PDF for more information.

Important: If you use the wsadmin tool to enable portlet fragment caching, you must make sure that servlet caching is also enabled. Similarly if you use the wsadmin tool to disable portlet fragment caching, you must make sure that servlet caching is also disabled. The settings for these two caching functions must stay synchronized. If you enable or disable portlet fragment caching using the administrative console, synchronization is automatically taken care of for you.

1. Locate the server object. The following example selects the first server found:

   Using Jacl:

   ```
   set s1 [$AdminConfig getid /Server:server1/]
   ```

   Using Jython:

   ```
   s1 = AdminConfig.getid('/Server:server1/')
   ```

2. List the Web containers and assign them to the `wc` variable, for example:

   Using Jacl:

   ```
   set wc [$AdminConfig list PortletContainer $s1]
   ```

   Using Jython:

   ```
   wc = AdminConfig.list('PortletContainer', s1)
   ```

3. Set the enablePortletCaching attribute to `true` and assign it to the `serEnable` variable, for example:

   Using Jacl:

   ```
   set serEnable "{enablePortletCaching true}"
   ```

   Using Jython:

   ```
   serEnable = [['enablePortletCaching', 'true']]
   ```

4. Enable caching, for example:

   Using Jacl:

   ```
   $AdminConfig modify $wc $serEnable
   ```

   Using Jython:

   ```
   AdminConfig.modify(wc, serEnable)
   ```

Configuring caching for Struts and Tiles applications

Use this task to cache Struts and Tiles applications.

Before you configure Struts and Tiles caching, you should have a developed application. For more information about developing Struts and Tiles applications, see The Apache Struts Web Application Framework on the Apache Web site at http://struts.apache.org/.

Use this task when you want to cache data in Struts and Tiles applications.

Struts is an open source framework for building Web applications using the Model-View-Controller (MVC) architecture. The Struts framework has a controller component and integrates with other technologies to provide the model and the view. Struts provide a control layer for the Web application, which reduces construction time and maintenance costs.

The Tiles framework builds on the jsp:include feature and is bundled with the Struts Web application framework. The Tiles framework reduces the duplication between JavaServer Pages (JSP) files and makes Web site layouts flexible and easy to maintain by assembling presentation pages from component parts.

Struts and Tiles caching is an extension of servlet and JSP caching, so the actions performed for each type of caching are very similar. See “Servlet caching” on page 1933 for more information.

1. Enable servlet and JSP caching. Enabling servlet caching automatically enables Struts and Tiles caching. See “Configuring servlet caching” on page 1933 for more information.
2. Develop the cache policy. A cache policy is required to cache a struts or tiles response.

   To develop a Struts cache policy:

   The Struts framework provides the controller component in the MVC-style application. The controller is a servlet called `org.apache.struts.action.ActionServlet.class`. In the `web.xml` file of the application, a servlet mapping of `*.do` is added for this Struts ActionServlet servlet so that every request for a Web address that ends with `.do` is processed. The ActionServlet servlet uses the information in the `struts-config.xml` file to decide which Struts action class runs the request for the specified resource.

   Cache policy using a previous version of WebSphere Application Server

   In the previous version of WebSphere Application Server, only one cache policy per servlet was supported. However, when you are using Struts, every request that ends in `.do` maps to the same ActionServlet servlet. To cache Struts responses, write a cache policy for the ActionServlet servlet based on its servlet path.

   For example, consider two Struts actions: `/HelloParam.do` and `/HelloAttr.do`. To cache the responses based on the id request parameter and the arg request attribute respectively, use the following cache policy:

```
<cache-entry>
 <class>servlet</class>
 <name>org.apache.struts.action.ActionServlet.class</name>
 <cache-id>
  <component id="" type="servletpath">
   <value>/HelloParm.do</value>
  </component>
 </cache-id>
 <cache-id>
  <component id="" type="servletpath">
   <value>/HelloAttr.do</value>
  </component>
  <component id="arg" type="attribute">
   <required>true</required>
  </component>
 </cache-id>
</cache-entry>
```

   Cache policy using WebSphere Application Server, Version 6.0 or later

With the current version of WebSphere Application Server, you can map multiple cache policies for a single servlet. You can rewrite the previous cache policy as in the following example:

```
<cache-entry>
 <class>servlet>
 <name>/HelloParam.do</name>
 <cache-id>
  <component id="id" type="parameter">
   <required>true</required>
  </component>
</cache-entry>
<cache-entry>
 <class>servlet</class>
 <name>/HelloAttr.do</name>
 <cache-id>
  <component id="arg" type="attribute">
   <required>true</required>
  </component>
 </cache-id>
</cache-entry>
```

To develop a Tiles cache policy:

The Tiles framework is built on the jsp:include tag, so everything that applies to JSP caching also applies to Tiles. You must set the flush attribute to `true` in any fragments that are included using the tiles:insert tag for the fragments to be cached correctly. The extra feature in tiles caching over JSP caching is based on the tiles attribute. For example, you might develop the following layout.jsp template:

```
<html>
 <%String categoryId = request.getParameter("categoryId")+"test"; %>
 <tiles:insert attribute="header">
  <tiles:put name="categoryId" value="<%= categoryId %>" />
 </tile:insert>
 <table>
  <tr>
   <td width="70%" valign="top"><tiles:insert attribute="body" /> </td>
  </tr>
  <tr>
   <td colspan="2"><tiles:insert attribute="footer" /></td>
  </tr>
 </table>
</body>
</html>
```

The nested tiles:put tag specifies the attribute of the inserted tile. In the `layout.jsp` template, the categoryId attribute is defined and passed on to the tile that is inserted into the placeholder for the header. In the following example, the `layout.jsp` file is inserted into another JSP file:

```
<html>
<body>
<tiles:insert page="layout.jsp?categoryId=1002" flush="true">
 <tiles:put name="header" value="/header.jsp" />
 <tiles:put name="body" value="/body.jsp" />
 <tiles:put name="footer" value="/footer.jsp" />
</tiles:insert>
</body>
</html>
```

The categoryId tile attribute is passed on to the `header.jsp` file. The `header.jsp` file can use the <tiles:useAttribute> tag to retrieve the value of categoryId. To cache the `header.jsp` file based on the value of the categoryId attribute, you can use the following cache policy:

```
<cache-entry>
 <class>servlet</class>
 <name>/header.jsp</name>
 <cache-id>
```

```
  <component id="categoryId" type="tiles_attribute">
  <required>true</required>
  </component>
 </cache-id>
</cache-entry>
```

3. Ensure your cache policy is working correctly. You can modify the policies within the `cachespec.xml` file while your application is running. See “Configuring cacheable objects with the cachespec.xml file” on page 1946 for more information about cache policies.

See “Task overview: Using the dynamic cache service to improve performance” on page 1913 for more information about the dynamic cache.

Configuring dynamic cache disk offload

Use this task to configure dynamic cache disk offload, which saves cache entries that are deleted from the memory cache to disk.

By default, when the number of cache entries reaches the configured limit for a given application server, cache entries are removed from the memory cache, allowing newer entries to be stored in the cache. Use disk offload to copy the cache entries that are being removed from the memory cache to disk for potential future access.

1. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Container services > Dynamic cache service**.
2. Select **Enable disk offload**.
3. After you enable the disk offload, you can set the **Disk offload location**. The disk offload location specifies where to save the cache entries on the disk. The disk offload location must be unique for any application servers that are defined on the same node. If you have multiple servers defined on the same node, make sure the disk offload location is different for each server.
4. Enable **Flush to disk** if you want cache objects that are in memory to be saved to disk when the server is stopped. Disk offload must be enabled if you choose this option. If you do not enable flush to disk, all the cache objects are deleted when the server stops.
5. Click **Apply** or **OK**.
6. Restart WebSphere Application Server.

You enabled disk offload. Memory cache entries are moved to disk for potential future access.

When you have two or more application servers with servlet caching enabled and the application servers specify the same disk offload location for their caches through the dynamic cache service, the following exceptions might occur:

```
java.lang.NullPointerException
       at com.ibm.ws.cache.CacheOnDisk.readTemplate(CacheOnDisk.java:686)
       at com.ibm.ws.cache.Cache.internalInvalidateByTemplate(Cache.java:828)
```

or:

```
java.lang.NullPointerException
       at com.ibm.ws.cache.CacheOnDisk.readCacheEntry(CacheOnDisk.java:600)
       at com.ibm.ws.cache.Cache.getCacheEntry(Cache.java:341)
```

If one server is run as root and the other servers are run as non-root, this problem could occur. For example, if `server1` runs as root and `server2` runs as `wasuser` or `wasgroup`, the cache files in the disk offload location might be created with root permissions. This situation causes the applications running on the non-root servers to crash when they try to read or write to the cache.

Managing cache entries stored on a disk:

Use this page to set Java virtual machine (JVM) custom properties to maintain cache entries that are saved to disk.

Steps for this task

You can set the custom properties globally to affect all cache instances, or you can set the custom property on a single cache instance. In most cases, set the properties on the individual cache instances. To set the custom properties on the default cache instance, use the global option. If you set the same property both globally and on a cache instance, the value that is set on the cache instance overrides the global value.

To configure the custom properties on a single object cache instance or servlet cache instance, perform the following steps:

1. In the administrative console, click one of the following paths:
   - To configure a servlet cache instance, click **Resources > Cache instances > Servlet cache instances >** *servlet_cache_instance_name* **> Custom properties > New**.
   - To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name* **> Custom properties > New**.
2. Type the name of the custom property. When configuring these custom properties on a single cache instance, you do not use the full property path. For example, type `explictBufferLimitOnStop` to configure the com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop custom property.
3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

To configure the custom property globally across all configured cache instances, perform the following steps:

1. In the administrative console, click **Servers > Application servers >** *server_name* **> Java and process management > Process definition > Java virtual machine > Custom properties > New**.
2. Type the name of the custom property (for example, com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop) in the **Name** field.
3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

com.ibm.ws.cache.CacheConfig.htodCleanupFrequency

Use this property to change the amount of time between disk cache cleanup.

Important: Setting this custom property manually is deprecated for V6.1. Therefore, you should use the administrative console to set this property. To set this property in the administrative console, click one of the following paths:

- To configure a servlet cache instance, click **Resources > Cache instances > Servlet cache instances >** *servlet_cache_instance_name*.
- To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field if it is not already selected.
2. Under Performance Settings, select Balanced performance and balanced memory usage or Custom.
3. In the Disk cache cleanup frequency field, specify an appropriate length of time, in minutes.

By default, the disk cache cleanup is scheduled to run at midnight to remove expired cache entries and cache entries that have not been accessed in the past 24 hours. However, if you have thousands of cache entries that might expire within one or two hours, the files that are in the disk cache can grow large and become unmanageable. Use the com.ibm.ws.cache.CacheConfig.htodCleanupFrequency custom property to change the time interval between disk cache cleanup.

Units	minutes For example, a value of 60 means 60 minutes between each disk cache cleanup.
Default	0 The disk cache cleanup occurs at midnight every 24 hours.

com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit

Use this property to specify the number of different cache IDs that can be saved in memory for the dependency ID and template buffers. Consider increasing this value if you have a lot of memory in your server and you want to increase the performance of your disk cache.

Important: Setting this custom property manually is deprecated for V6.1. Therefore, you should use the administrative console to set this property. To set this property in the administrative console, click one of the following paths:

- To configure a servlet cache instance, click **Resources > Cache instances > Servlet cache instances >** *servlet_cache_instance_name*.
- To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field, if it is not already selected.
2. Under Disk Cache settings, select Limit disk cache size in entries, if it is not already selected.
3. In the Disk cache size field, specify the number of cache IDs that can be saved in memory for the dependency ID and template buffers.

Units	number of cache IDs For example, a value of 1000 means that each dependency ID or template ID can have up to 1000 different cache IDs in memory.
Default	1000
Minimum	100

Tune the delay offload function

Use these properties to tune the delay offload function for the disk cache.

Important: Setting these custom properties manually is deprecated for V6.1. You should use the administrative console to set these properties. The individual property descriptions include information on how to use the administrative console to set these properties.

The delay offload function uses extra memory buffers for dependency IDs and templates to delay the disk offload and minimize the input and output operations. However, if most of your cache IDs are longer than 100 bytes, the delay offload function might use too much memory. Use any combination of the following properties to tune your configuration:

- To increase or decrease the in-memory limit of cache IDs for dependency ID and template buffers, use the com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit custom property.
- To disable the disk cache delay offload function, use the com.ibm.ws.cache.CacheConfig.htodDelayOffload custom property. Disabling this property saves all cache entries to disk immediately after removing them from the memory cache.

com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop

Use this custom property when the flush-to-disk-on-stop feature is enabled. When the server is stopping, offloads are limited to the value specified for this property, pending removal of entries in the explicit invalidation buffer. If this property is set to `0`, there is no limit to the number of offloads that can occur. Only positive integers are accepted as values for this property. If the number of entries in the explicit invalidation buffer is greater then the specified limit, all of the disk files for this specified cache instance are deleted after the server stops.

Important: You cannot use the administrative console to set this property.

Configuring Edge Side Include caching

The Web server plug-in contains a built-in ESI processor. The ESI processor can cache whole pages, as well as fragments, providing a higher cache hit ratio. The cache implemented by the ESI processor is an in-memory cache, not a disk cache, therefore, the cache entries are not saved when the Web server is restarted.

Edge Side Include (ESI) is configured through the `plugin-cfg.xml` file.

When a request is received by the Web server plug-in, it is sent to the ESI processor, unless the ESI processor is disabled. It is enabled by default. If a cache miss occurs, a Surrogate-Capabilities header is added to the request and the request is forwarded to the WebSphere Application Server. If servlet caching is enabled in the application server, and the response is edge cacheable, the application server returns a Surrogate-Control header in response to the WebSphere Application Server plug-in.

The value of the Surrogate-Control response header contains the list of rules that are used by the ESI processor to generate the cache ID. The response is then stored in the ESI cache, using the cache ID as the key. For each ESI include tag in the body of the response, a new request is processed so that each nested include results in either a cache hit or another request that forwards to the application server. When all nested includes have been processed, the page is assembled and returned to the client.

The ESI processor is configurable through the WebSphere Web server plug-in configuration file `plugin-cfg.xml`. The following is an example of the beginning of this file, which illustrates the ESI configuration options.

```
<?xml version-"1.0"?>
<Config>
  <Property Name="esiEnable" Value="true"/>
  <Property Name="esiMaxCacheSize" Value="1024"/>
  <Property Name="esiInvalidationMonitor" Value="false"/>
```

- The first option, esiEnable, can be used to disable the ESI processor by setting the value to false. ESI is enabled by default. If ESI is disabled, then the other ESI options are ignored.
- The second option, esiMaxCacheSize, is the maximum size of the cache in 1K byte units. The default maximum size of the cache is 1 megabyte. If the cache is full, the first entry to be evicted from the cache is the entry that is closest to expiration.

- The third option, esiInvalidationMonitor, specifies if the ESI processor should receive invalidations from the application server. ESI works well when the Web servers following a threading model are used, and only one process is started. When multiple processes are started, each process caches the responses independently and the cache is not shared. This could lead to a situation where, the system's memory is fully used up by ESI processor. There are three methods by which entries are removed from the ESI cache: first, an entry expiration timeout occurs; second, an entry is purged to make room for newer entries; or third, the application server sends an explicit invalidation for a group of entries. For the third mechanism to be enabled, the esiInvalidationMonitor property must be set to true and the DynaCacheEsi application must be installed on the application server. The DynaCacheEsi application is located in the installableApps directory and is named DynaCacheEsi.ear. If the ESIInvalidationMonitor property is set to true but the DynaCacheEsi application is not installed, then errors occur in the Web server plug-in and the request fails.
- On distributed platforms, the cache for the ESI processor is monitored through the CacheMonitor application. In order for ESI processor cache to be visible in the CacheMonitor, the DynaCacheEsi application must be installed as described above, and the ESIInvalidationMonitor property must be set to true in the `plugin-cfg.xml` file.
- When WebSphere Application Server is used to serve static data, such as images and HTML on the application server, the URLs are also cached in the ESI processor. This data has a default timeout of 300 seconds. You can change the timeout value by adding the property com.ibm.servlet.file.esi.timeOut to the Java virtual machine (JVM) command line parameters. The following example shows how to set a one minute timeout on static data cached in the plug-in:

  ```
  -Dcom.ibm.servlet.file.esi.timeOut=60
  ```

  For information about configuring alternate URL, see the *Tuning guide* PDF.

Configuring alternate URL:

Alternate URL is a method for edge caching JavaServer Pages (JSP) files and servlet responses that you can not request externally. Dynamic cache provides support to recognize the presence of an Edge Side Include (ESI) processor and to generate ESI include tags and appropriate cache policies for edge fragments that can be cached. However, you must be able to externally request an edge fragment from the application server before it can be cached. In other words, if a user types the URL in their browser with the appropriate parameters and cookies for the fragment, WebSphere Application Server must be able to return the content for that fragment.

One of the standard Java 2 Platform, Enterprise Edition (J2EE) programming architectures is the model-view-controller (MVC) architecture, where a call to a controller servlet might include one or more child JSP files to construct the view. When using the MVC programming model, the child JSP files are edge cached only if you can request these JSP files externally, which is not usually the case. For example, if a child JSP file uses one or more request attributes that are determined and set by the controller servlet, you cannot cache that JSP file on the edge. You can use alternate URL support to overcome this limitation by providing an alternate controller servlet URL used to invoke the JSP file.

The alternate URL for a JSP file or a servlet is set in the `cachespec.xml` file as a property with the name `alternate_url`. You can set the alternate URL either on a per cache-entry basis or on a per cache-id basis. It is valid only if the `EdgeCacheable` property is also set for that entry. If the `EdgeCacheable` property is not set, the `alternate_url` property is ignored. The following is a sample cache policy using the `alternate_url` property:

```
<cache-entry>
    <class>servlet</class>
    <name>/AltUrlTest2.jsp</name>
    <property  name="EdgeCacheable">true</property>
    <property  name="alternate_url">/alturlcontroller2</property>
        <cache-id>
            <timeout>600</timeout>
```

```
            <priority>2</priority>
        </cache-id>
</cache-entry>
```

For more information on the `cachespec.xml` file, see “Cachespec.xml file” on page 1947.

Configuring external cache groups

The dynamic cache can control caches outside of the application server, such as the Edge server, an IBM HTTP Server, or an HTTP Server ESI Fragment Processor plug-in.

When external cache groups are defined, the dynamic cache matches externally cacheable cache entries with those groups, and pushes cache entries and invalidations out to those groups. This allows WebSphere Application Server to manage dynamic content beyond the application server. The content can then be served from the external cache, instead of the application server, improving savings in performance.

1. Open the administrative console.
2. Enable the dynamic cache.
   a. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Container services > Dynamic cache service**.
   b. Select **Enable service at server startup** to enable the dynamic cache each time the application server starts.
3. Define the external cache group that WebSphere Application Server should control.
   a. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Container services > Dynamic cache service > External cache groups**.
   b. Click **New** or choose an external cache group from the list.
4. Configure cache group members.
   a. Click **External cache groups** from the dynamic cache administrative console page. Then click **New** or choose an external cache group from the list.
   b. Click **External cache group members** > **New** or choose an external cache group member from the list.
   c. Type the configuration string in the **Address** field.
   d. Type the adapter bean name in the **Adapter Bean Name** field.
   e. **Save** the configuration.
   f. Click **Apply** or **OK**.

External cache group collection:

Use this page to define sets of external caches controlled by WebSphere Application Server on Web servers such as IBM Edge Server and IBM HTTP Server.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Container services > Dynamic cache service** > **External cache groups**.

Name:

Specifies the external cache group name.

The external cache group name needs to match the **ExternalCache** property as defined in the servlet or JavaServer Pages file `cachespec.xml` file.

When external caching is enabled, the cache matches pages with its Universal Resource Identifiers (URI) and pushes matching pages to the external cache. The entries can then be served from the external cache, instead of from the application server.

Type:

Specifies the external cache group type.

External cache group settings:

Use this page to configure sets of external caches controlled by WebSphere Application Server on Web servers, such as IBM Edge Server and IBM HTTP Server.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Container services > Dynamic cache service** > **External cache groups** > *external_cache_group*.

Name:

Specifies the external cache group name.

The external cache group name must match the **ExternalCache** property as defined in the servlet or JavaServer Pages (JSPs) `cachespec.xml` file.

When external caching is enabled, the cache matches pages with its Universal Resource Identifiers (URIs) and pushes matching pages to the external cache. The entries can then be served from the external cache, instead of the application server. This ability creates a significant savings in performance.

External cache group member collection:

Use this page to define specific caches that are members of a cache group.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Container services > Dynamic cache service** > **External cache groups** > *external_cache_group* > **External cache group members**.

Address:

Specifies a configuration string used by external cache adapter bean to connect to the external cache.

AdapterBeanName:

Specifies the adapter bean name.

Example adapter bean names supported in WebSphere Application Server are:

AFPA
AdapterBeanName: com.ibm.ws.cache.servlet.Afpa
Address: Port on which afpa listens
ESI
AdapterBeanName: com.ibm.websphere.servlet.cache.ESIInvalidatorServlet
Address: local host
IBM Web Traffic Express (WTE) (IBM Edge Server)
AdapterBeanName: com.ibm.websphere.edge.dynacache.WteAdapter
Address: hostname:port (host name and port on which WTE is listening)

External cache group member settings:

Use this page to configure specific caches that are members of a cache group.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Container services > Dynamic cache service** > **External cache groups** > *external_cache_group* > **External cache group members** > *external_cache_group_member*.

Address:

Specifies a configuration string used by external cache adapter bean to connect to the external cache.

Adapter bean name:

Specifies the adapter bean name.

Example adapter bean names supported in WebSphere Application Server are:

AFPA
AdapterBeanName: com.ibm.ws.cache.servlet.Afpa
Address: Port on which afpa listens
ESI
AdapterBeanName: com.ibm.websphere.servlet.cache.ESIInvalidatorServlet
Address: local host
IBM Web Traffic Express (WTE) (IBM Edge Server)
AdapterBeanName: com.ibm.websphere.edge.dynacache.WteAdapter
Address: hostname:port (host name and port on which WTE is listening)

Configuring high-speed external caching through the Web server:

IBM HTTP Server for Windows NT and Windows 2000 operating systems contains a high-speed cache referred to as the *Fast Response Cache Accelerator*, or *cache accelerator*. The Fast Response Cache Accelerator is available on Windows NT and Windows 2000 operating systems and AIX platforms. However, support to cache dynamic content is only available on Windows NT and Windows 2000 operating systems. You can enable cache accelerator to cache static and dynamic content.

To enable cache accelerator for caching static content, add the following directives to the `http.conf` configuration file, in the IBM HTTP Server `conf` directory:

- `AfpaEnable`
- `AfpaCache on`
- `AfpaLogFile "app_server_root\IBMHttpServer\logs\afpalog" V-ECLF`

To enable cache accelerator for caching dynamic content, such as servlets and JavaServer Pages (JSP) files, configure the WebSphere Application Server and the IBM HTTP Server for distributed platforms:

1. Configure WebSphere Application Server to enable Fast Response Cache Accelerator. It is important to follow all the steps for every application server in the cluster.
   a. Turn on servlet caching for each application server that uses the cache accelerator.
   b. Configure an external cache group on the application server:
      1) Click **Servers > Application servers >** *server_name* **> Container services > Dynamic cache service > External cache groups**.
      2) Click **New** on the External cache group administrative console page to define an external cache group named `afpa` for each application server that uses the cache accelerator.
      3) In the **External cache group** field, type `afpa` and apply the changes.
   c. Add a member to the group with an adapter bean name of `com.ibm.ws.cache.servlet.Afpa`.

1) Click **Afpa > External cache group members**.
2) Click **New** on the External cache group members administrative console page.
3) In the **AdapterBean name** field, type `com.ibm.ws.cache.servlet.Afpa`.
4) In the **Address** field, enter an unused port number.

d. Add a cache policy in the `cachespec.xml` file for the servlet or JSP file you want to cache. Add the following property to the cache policy:

```
<property name="ExternalCache">afpa</property>
```

2. Enable cache accelerator on the IBM HTTP Server for distributed platforms:

a. Add the following directives to the end of the `httpd.conf` file:
- `AfpaEnable`
- `AfpaCache on`
- `AfpaLogFile "app_server_root\IBMHttpServer\logs\afpalog" V-ECLF`
- `LoadModule afpaplugin_module app_server_root/bin/afpaplugin.dll`
- `AfpaPluginHost WAS_Hostname:port`, where *WAS_Hostname* is the host name of the application server and *port* is the port you specified in the Address field while configuring the external cache group member

The LoadModule directive loads the IBM HTTP Server plug-in that connects the Fast Response Cache Accelerator to the WebSphere Application Server fragment cache. If multiple IBM HTTP Servers are routing requests to a single application server, add the directives above to the `http.conf` file of each of these IBM HTTP Servers for distributed platforms. If one IBM HTTP Server is routing requests to a cluster of application servers, add the `AfpaPluginHost WAS_Hostname:port` directive to the `http.conf` file for each application server in the cluster. For example, if there are three application servers in the cluster, add the following directives to the `http.conf` file:
- `LoadModule afpaplugin_module app_server_root/bin/afpaplugin.dll`
- `AfpaPluginHost WAS1_Hostname:port1`
- `AfpaPluginHost WAS2_Hostname:port2`
- `AfpaPluginHost WAS3_Hostname:port3`

Configuring fast response cache accelerator cache size through a distributed platforms Web server:

In the default IBM HTTP Server for distributed platforms configuration, the maximum fast cache accelerator dynamic cache size is calculated as 1/8 of physical pin-able memory.

On a machine with 384 megabytes of RAM, it allows a maximum of approximately 50 megabytes for the Fast Cache Accelerator dynamic cache. When this limit is reached, the cache accelerator deletes older entries to cache new entries.

Using the IBM HTTP Server for distributed platforms AfpaDynaCacheMax directive, tune the maximum allowed cache size:

1. Place the directive in the global server configuration scope, along with the other default Fast Cache Accelerator directives.
2. Enable fast cache accelerator. To enable the fast cache accelerator, update the following directives in this IBM HTTP Server's `http.conf` file:

```
AfpaEnable
AfpaCache on
AfpaLogFile "c:/Program Files/IBM HTTP Server/logs/afpalog" V-ECLF
AfpaDynaCacheMax 10
```

These settings limit the dynamic cache size to 10 megabytes. If you use these directives to increase cache size, do not make the cache so large that all the physical memory is consumed. Determine how much memory is available when all applications are running, by using the Windows Task Manager.

Assign no more than 50% of available physical memory to the dynamic cache. Specifying too large a cache not only decreases the performance of other applications, but also puts you at a risk for completely running out of memory.

The default configuration does not include the AfpaDynaCacheMax directive where the cache size is automatically calculated as 1/8 of physical memory.

Configuring cacheable objects with the cachespec.xml file

Use this task to define cacheable objects inside the `cachespec.xml`, found inside the Web module `WEB-INF` or enterprise bean `META-INF` directory.

Enable the dynamic cache. See “Enabling the dynamic cache service” on page 1929 for more information.

You can save a global `cachespec.xml` in the application server properties directory, but the recommended method is to place the cache configuration file with the deployment module. The root element of the `cachespec.xml` file is <cache>, which contains <cache-entry> elements.

The <cache-entry> element can be nested within the <cache> element or a <cache-instance> element. The <cache-entry> elements that are nested within the <cache> element are cached in the default cache instance. Any <cache-entry> elements that are in the <cache-instance> element are cached in the instance that is specified in the **name** attribute on the <cache-instance> element.

Within a <cache-entry> element are parameters that allow you to complete the following tasks to enable the dynamic cache with the `cachespec.xml` file:

1. Develop a `cachespec.xml` file.
   a. Create a caching configuration file.

      In the *<app_server_root>*`/properties` directory, locate the `cachespec.sample.xml` file.

   b. Copy the `cachespec.sample.xml` file to `cachespec.xml` in Web module `WEB-INF` or enterprise bean `META-INF` directory.
2. Define the cache-entry elements necessary to identify the cacheable objects. See the topic “Cachespec.xml file” on page 1947 for a list of elements.
3. Develop cache ID rules.

   To cache an object, WebSphere Application Server must know how to generate unique IDs for different invocations of that object. The *<cache-id>* element performs that task. Each cache entry can have multiple cache-ID rules that run in order until either a rule returns cache-ID that is not empty or no more rules remain to run. If no cache-ID generation rules produce a valid cache ID, then the object is not cached. Develop the cache IDs in one of two ways:

   - Use the *<component>* element defined in the cache policy of a cache entry (recommended). See “Cachespec.xml file” on page 1947 for more information about the *<component>* element.
   - Write custom Java code to build the ID from input variables and system state. To configure the cache entry to use the ID generator, specify your IdGenerator in the XML file by using the *<idgenerator>* tag, for example:

```
<cache-entry>
     <class>servlet</class>
     <name>/servlet/CommandProcessor</name>
  <cache-id>
      <idgenerator>com.mycompany.SampleIdGeneratorImpl</idgenerator>
      <timeout>60</timeout>
  </cache-id>
</cache-entry>
```

4. Specify dependency ID rules. Use dependency ID elements to specify additional cache group identifiers that associate multiple cache entries to the same group identifier.

   The dependency ID is generated by concatenating the dependency ID base string with the values returned by its component elements. If a required component returns a null value, then the entire dependency ID does not generate and is not used. You can validate the dependency IDs explicitly through the dynamic cache API, or use another cache-entry *<invalidation>* element. Multiple dependency ID rules can exist per cache entry. All dependency ID rules run separately. See “Cachespec.xml file” on page 1947 for a list of *<component>* elements.

5. Invalidate other cache entries as a side effect of this object start, if relevant. You can define invalidation rules in exactly the same manner as dependency IDs. However, the IDs that are generated by invalidation rules are used to invalidate cache entries that have those same dependency IDs.

   The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules run separately.
6. Ensure your cache policy is working correctly. You can modify the policies within the `cachespec.xml` file while your application is running. The dynamic cache reloads the updated file automatically. If you are caching static content and you are adding the cache policy to an application for the first time, you must restart the application. You do not need to restart the application server to activate the new cache policy. See "Verifying the cacheable page" for more information.

Typically you declare several *<cache-entry>* elements inside a `cachespec.xml` file.

When new versions of the `cachespec.xml` are detected, the old policies are replaced. Objects that cached through the old policy file are not automatically invalidated from the cache; they are either reused with the new policy or eliminated from the cache through its replacement algorithm.

For each of the three IDs (cache, dependency, invalidation) generated by cache entries, a *<cache-entry>* can contain multiple elements. The dynamic cache runs the *<cache-id>* rules in order, and the first one that successfully generates an ID is used to cache that output. If the object is to be cached, each one of the *<dependency-id>* elements is run to build a set of dependency IDs for that cache entry. Finally, each of the *<invalidation>* elements are run, building a list of IDs that the dynamic cache invalidates, whether or not this object is cached.

Verifying the cacheable page

Use this task to verify that the dynamic cache service has its cache policies configured correctly and is serving cached content.

The dynamic cache service should be enabled. You should have a cache policy developed for your application. See "Configuring cacheable objects with the cachespec.xml file" on page 1946 for more information. You must have servlet caching enabled in the web container. See "Configuring servlet caching" on page 1933 for more information.

You can verify the cacheable page by invoking the snoop servlet in the default application. If the dynamic cache is working correctly, refreshing the servlet repeatedly results in viewing cached content.

1. View the Snoop servlet in the default application by accessing the URI: `/snoop` The Snoop servlet is a part of the default application. See "Default Application" on page 103 for more information.
2. Invoke and reload the URI several times using a different Web browser or using different parameters. This action returns the same output for the snoop servlet. The snoop servlet is now operating incorrectly, because it displays the request information from its first invocation rather than from the current request.
3. Inspect the entry in the cache with the dynamic cache monitor. See "Displaying cache information" on page 1977 for more information.

Cachespec.xml file

The cache parses the `cachespec.xml` file when the server starts, and extracts a set of configuration parameters from each cache-entry element. Every time a new servlet or other cacheable object initializes, the cache attempts to match each of the cache-entry elements to find the configuration information for that object.

The cache-entry elements can be inside the root cache element or inside a cache-instance element. Cache entries that are in the root element are cached with the default cache instance. Cache entries that

are in the <cache-instance> element are cached in that particular cache instance. Different cacheable objects have different class elements. You can define the specific object that a cache policy refers to using the name element.

Location

Place the `cachespec.xml` file with the deployment module. Use an assembly tool to define the cacheable objects. See Assembling applications for more information about assembling applications. You can also place a global `cachespec.xml` file in the application server properties directory.

The `cachespec.dtd` file is available in the application server properties directory. The `cachespec.dtd` file defines the legal structure and the elements that can be in your `cachespec.xml` file.

Usage notes

Cachespec.xml elements

The root element of the `cachespec.xml` file is cache and contains cache-instance and cache-entry elements. The cache-entry elements can also be placed inside of cache-instance elements to make that cache entry part of a cache instance that is different from the default.

cache-instance

```
<cache-instance name="cache_instance_name"></cache-instance>
```

The name attribute is the Java Naming and Directory Interface (JNDI) name of the cache instance that is set in the administrative console.

Each cache-instance element must contain at least one cache-entry element. A cache entry that is matched within a cache-instance element is cached in the servlet cache instance that is specified by the name attribute. If identical cache-entry elements exist across cache-instance elements, the first cache-entry element that is matched is used.

cache-entry

Each cache entry must specify certain basic information that the dynamic cache uses to process that entry. This section explains the function of each cache entry element of the `cachespec.xml` file including:

- class
- name
- sharing-policy
- skip-cache
- property
- cache-id

With the current version of WebSphere Application Server, you can define multiple cache policies for a single servlet. For example, if you define multiple mappings for a servlet in the `web.xml` file, you can create a cache entry for each one of the mappings.

class

```
<class>command | servlet | webservice | JAXRPCClient | static | portlet </class>
```

This element is required and specifies how the application server interprets the remaining cache policy definition. The value `servlet` refers to servlets and JavaServer Pages (JSP) files that are deployed in the WebSphere Application Server servlet engine. The `webservice` class extends the servlet with special component types for Web services requests. The `JAXRPCClient` is used to define a cache entry for the

Web services client cache. The value, command, refers to classes using the WebSphere Application Server command programming model. The value, static, refers to files that contain static content. The following examples illustrate the class element:

```
<class>command</class>
<class>servlet</class>
<class>webservice</class>
<class>JAXRPCClient</class>
<class>static</class>
<class>portlet</class>
```

name

```
<name>name</name>
```

Use the following guidelines for the name element to specify a cacheable object:

- For commands, this required element must include the package name, if any, and class name, including a trailing .class, of the configured object.
- For servlets and JSP files, if the cachespec.xml file is in the WebSphere Application Server properties directory, this required element must include the full URI of the JSP file or servlet to cache. For servlets and JSP files, if the cachespec.xml file is in the Web application, this required element can be relative to the specific Web application context root.
- For Web services, include the Universal Resource Identifier (URI) of the Simple Object Access Protocol (SOAP) router that is associated with the Web service that you want to cache.
- For Web services client cache, the name is the target end point of the cacheable Web service or the URI of the SOAP router that is associated with the cacheable Web service. You can use the SOAP address location in the Web Services Description Language (WSDL) file to define the name for the Web services client cache.
- For static files, if the cachespec.xml file is in the WebSphere Application Server properties directory, this required element must include the full URI of the file to cache. If the cachespec.xml file is in the Web application, this required element can be relative to the specific Web application context root. For a Web application with a context root, the cache policy for files using the static class must be specified in the Web application, and not in the properties directory.
- For portlets, if the cachespec.xml file is in the WebSphere Application Server properties directory, this required element must include the full context path and name of the portlet to cache. If the cachespec.xml file is in the Web application, this required element is the portlet name that is relative to the specific Web application context root.

Tip: The preferred location of the cachespec.xml file is in the Web application, not the properties directory.

You can specify multiple name elements within a cache-entry if you have different mappings that refer to the same servlet.

The following examples illustrate the name element:

```
<name>com.mycompany.MyCommand.class</name>
<name>default_host:/servlet/snoop</name>
<name>com.mycompany.beans.MyJavaBean</name>
<name>mywebapp/myjsp.jsp</name>
<name>/soap/servlet/soaprouter</name>
<name>http://remotecompany.com:9080/service/getquote</name>
<name>mywebapp/myLogo.gif</name>
```

sharing-policy

```
<sharing-policy> not-shared | shared-push | shared-pull | shared-push-pull</sharing-policy>
```

When working within a cluster with a distributed cache, these values determine the sharing characteristics of entries that are created from this object. If this element is not present, a not-shared value is assumed.

In single server environments, not-shared is the only valid value. When enabling a replication, the default value is `not-shared` . This property does not affect distribution to Edge Side Include processors through the Edge fragment caching property. See for more information.

Value	Description
not-shared	Cache entries for this object are not shared among different application servers. These entries can contain non-serializable data. For example, a cached servlet can place non-serializable objects into the request attributes, if the <class> type supports it.
shared-push	Cache entries for this object are automatically distributed to the dynamic caches in other application servers or cooperating Java virtual machines (JVMs). Each cache has a copy of the entry at the time it is created. These entries cannot store non-serializable data.
shared-pull	Cache entries for this object are shared between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server runs the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended.
shared-push-pull	Cache entries for this object are shared between application servers on demand. When an application server generates a cache entry, it broadcasts the cache ID of the created entry to all cooperating application servers. Each server then knows whether an entry exists for any given cache ID. On a given request for that entry, the application server knows whether to generate the entry or pull it from somewhere else. These entries cannot store non-serializable data.

The following example shows a sharing policy:

```
<sharing-policy>not-shared</sharing-policy>
```

skip-cache

Takes the name of a request attribute, which if present in the request context, dictates that the response cannot be retrieved from the cache instance that is specified. This property is useful for previewing content in production systems and verifying that the application is working and performing as expected.

```
<cache>
 <skip-cache-attribute>att1</skip-cache-attribute> <!—Applies only to the base cache- ->
...
<cache-instance name="instance1">
<skip-cache-attribute>att2</skip-cache-attribute> <!-Applies only to this instance- ->
...
</cache-instance>
</cache>
```

property

```
<property name="key">value</property>
```

where *key* is the name of the property for this cache entry element, and *value* is the corresponding value.

You can set optional properties on a cacheable object, such as a description of the configured servlet. The class determines valid properties of the cache entry. At this time, the following properties are defined:

Property	Valid classes	Value
ApplicationName	All	Overrides the J2EEName application ID so that multiple applications can share a common cache ID namespace.
EdgeCacheable	Servlet	True or false. The default is `false`. If the property is `true`, then the given servlet or JSP file is externally requested from an Edge Side Include processor. Whether or not the servlet or JSP file is cacheable depends on the rest of the cache specification.
ExternalCache	Servlet and portlet	Specifies the external cache name. The external cache name needs to match the external cache group name.
consume-subfragments	Servlet, Web service, or portlet	True or false. The default is `false`. When a servlet is cached, only the content of that servlet is stored, and includes placeholders for any other fragments to which it includes or forwards. Consume-subfragments (CSF) tells the cache not to stop saving content when it includes a child servlet. The parent entry, the one marked CSF, includes all the content from all fragments in its cache entry, resulting in one big cache entry that has no includes or forwards, but the content from the whole tree of entries. Consume-subfragments can save a significant amount of application server processing, but is typically only useful when the external HTTP request contains all the information needed to determine the entire tree of included fragments. Use the <exclude> element to tell the cache to stop consuming for the excluded fragment and instead, create a placeholder for the include or forward. For example, exclude `A.jsp` from the consume-subfragment, as follows: `<property name= "consume-sbufragments">true <exclude>/A.jsp<exclude> </property>`

do-not-consume	Servlet, Web service, or portlet	True or false. The default is `false`. When a fragment parent has the consume-subfragment property set to true the child fragment content is saved in the cache entry of the parent. Do-not-consume (DNC) tells the cache to stop saving the content for this fragment in the parent cache-entry and create a placeholder instead for the include or forward.
alternate_url	Servlet	Specifies the alternate URL that is used to invoke the servlet or JSP file. The property is valid only if the EdgeCacheable property also is set for the cache entry.
persist-to-disk	All	True or false. The default is `true`. When this property is set to `false`, the cache entry is not written to the disk when overflow or server stopping occurs.
save-attributes	Servlet and portlet	True or false. The default is `true`. When this property is set to `false`, the request attributes are not saved with the cache entry. Use the <exclude> element to specify the request attributes that do not apply to the save-attributes property. For example, to save only the attr1 attribute with the cache entry: `<property name="save-attributes">false` `<exclude>attr1</exclude>` `</property>` To save all attributes except the `attr1` attribute in the cache entry, set the property to true in the preceding sample. If you do not use the <exclude> element, either all or no request attributes are saved with the cache entry.
delay-invalidations	Command	True or false. When this property is set to `true`, the commands that are invalidating cached objects based on the invalidation rules in this cache entry invalidate the cache entries after running. By default, the invalidation occurs before the command runs.

store-cookies	Servlet and portlet	Takes one or more cookie name as its argument which is saved along with the cache object and restored by the servlet cache in the response with a set-cookie header. Save all cookies except cookie1 as part of the cache-entry as follows: `<property name= "store-cookies">true <exclude>cookie</exclude> </property>` Save only cookie1 as part of the cache-entry, as follows: `<property name= "store-cookies">false <exclude><cookie1</exclude> </property>`
ignore-get-post	Servlet and portlet	True or false. The default is `false`. When the property is set to `true` the request type is not appended to the cache-id for GET and POST requests unless the requestType component requestType component subelement is defined. By default the request type is automatically appended to the cache-id for GET and POST requests.
do-not-cache	Servlet and portlet	Defines a fragment that is neither cached nor consumed by its parent. `<cache-entry> ... <property name="do-not-cache"> true</property>` or `<cache-id> <property name="do-not-cache"> true</property> </cache-id> </cache-entry>`

cache-id

To cache an object, the application server must know how to generate a unique ID for different invocations of that object. These IDs are built either from user-written custom Java code or from rules that are defined in the cache policy of each cache entry. Each cache entry can have multiple cache ID rules that run in order until either:

- A rule returns a non-empty cache ID, or
- No more rules are left to run.

If none of the cache ID generation rules produce a valid cache ID, the object is not cached.

Each cache-id element defines a rule for caching an object and is composed of the sub-elements component, timeout, inactivity, priority, property, idgenerator, and metadatagenerator. The following example illustrates a cache-id element:

```
<cache-id>
   component*| timeout? | inactivity? | priority? | property* | idgenerator? | metadatagenerator?
</cache-id>
```

component subelement

Use the component subelement to generate a portion of the cache ID. The component subelement consists of the attributes id, type, and ignore-value, and the elements index, method, field, required, value, and not-value.

- Use the id attribute to identify the component.
- Use the type attribute to identify the type of component. The following table lists the values for the type.

Type	Valid classes	Meaning
method	Command	Calls the indicated method on the command or object
field	Command	Retrieves the named field in the command or object
parameter	Servlet and portlet	Retrieves the named parameter value from the request object
parameter-list	Servlet and portlet	Retrieves a list of values for the named parameter
session	Servlet and portlet	Retrieves the named value from the HTTP session
cookie	Servlet	Retrieves the named cookie value
attribute	Servlet	Retrieves the named request attribute
header	Servlet, Web service, and portlet	Retrieves the named request header
pathInfo	Servlet	Retrieves the pathInfo element from the request
servletpath	Servlet	Retrieves the servlet path
locale	Servlet and portlet	Retrieves the request locale
requestType	Servlet and portlet	Retrieves the HTTP request method from the request.
tiles_attribute	Servlet and portlet	Retrieves the value of an attribute from a tile.
SOAPEnvelope	Web service and Web services client cache	Retrieves the SOAPEnvelope element from a Web services request. An ID attribute of Hash uses a Hash of the SOAPEnvelope element, while Literal uses the SOAPEnvelope element as received.
SOAPAction	Web service	Retrieves the SOAPAction header, if available, for a Web services request.
serviceOperation	Web service	Retrieves the service operation for a Web services request
serviceOperationParameter	Web service	Retrieves the specified parameter from a Web services request

Type	Valid classes	Meaning
operation	Web services client cache	Indicates an operation type in the Web Services Description Language (WSDL) file. The id attribute is ignored and the value is the operation or method name. If the namespace of the operation is specified, format the value as namespaceOfOperation:nameOfOperation
part	Web services client cache	Indicates an input message part in the WSDL file or a request parameter. Its id attribute is the part or parameter name, and the value is the part or parameter value.
SOAPHeaderEntry	Web services client cache	Retrieves special information in the Simple Object Access Protocol (SOAP) header of the Web services request. The id attribute specifies the name of the entry. In addition, the entry of the SOAP header in the SOAP request must have the actor attribute, which contains com.ibm.websphere.cache. For example: `<soapenv:Header>` `<getQuote soapenv:actor=` `"com.ibm.websphere.cache">IBM` `</getQuote>` `</soapenv:Header>`
portletSession	Portlet	Retrieves the named value from the portlet session
portletWindowId	Portlet	Retrieves the portlet window ID from the portlet request object
portletMode	Portlet	Retrieves the portlet mode from the portlet request object
portletWindowsState	Portlet	Retrieves the portlet window state from the portlet request object
sessionID	Servlet and portlet	Retrieves the HTTP session ID

- Use the ignore-value attribute to specify whether or not to use the value that is returned by this component in cache ID formation. This attribute is optional with a default value of `false`. If the value is `true`, only the ID of the component is used when creating a cache ID, or no output is used when creating a dependency or invalidation ID.
- Use the method element to call a void method on a returned object. You can infinitely nest method and field objects in any combination. The method must be public and is not valid for edge-cacheable components. For example:

  ```
  <component id="getUser" type="method"><method>getUserInfo
  <method>getName</method></method></component>
  ```

  This method is equivalent to getUser().getUserInfo().getName()

  For component types attribute, method, or field that can return an object, when the object returned is a collection or array, the ID is created with a comma separated list of the elements in the collection or array. For example, if the request attribute users returns an array [a, b] and the cache entry is defined like the following example:

```
<cache-entry>
 <class>servlet</class>
 <name>xxx.jsp</name>
 <cache-id>
  .
  .
  <component id="users" type="attribute">
   <required>true</required>
  </component>
  .
  .
 </cache-id>
 <dependency-id>dep
  <component id="users" type="attribute">
   <required>true</required>
  </component>
 </dependency-id>
</cache-entry>
```

The cache id contains the string `users: a,b`. The dependency id is `dep: a,b`.

Use the multipleIDs attribute with the component types to specify and generate multiple dependency IDs (or invalidation IDs), based on the items in the collection or array. For example:

```
<cache-entry>
 <class>servlet</class>
 <name>xxx.jsp</name>
 <cache-id>
  .
  .
   <component id="users" type="attribute">
    <required>true</required>
   </component>
  .
  .
 </cache-id>
 <dependency-id>dep
  <component id="users" type="attribute" multipleIDs="true">
   <required>true</required>
  </component>
 </dependency-id>
</cache-entry>
```

The cache policy will generates the following dependency IDs:

- `dep:a,b`
- `dep:a`
- `dep:b`

Use the index element with the prvious component type to add only the value of the element at the specified index position in the collection or array, to the ID that is being created.

```
<cache-entry>
 <class>servlet</class>
 <name>xxx.jsp</name>
 <cache-id>
  .
  .
   <component id="users" type="attribute">
    <required>true</required>
    <index>1</index>
   </component>
  .
  .
 </cache-id>
 <dependency-id>dep
  <component id="users" type="attribute" multipleIDs="true">
```

```
   <required>true</required>
  </component>
 </dependency-id>
</cache-entry>
```

The previous cache policy generates the following component to use in the cache ID: `users: b`. Use the <method> element to call a void method on a returned object.

- Use the field element to access a field in a returned object. You can infinitely nest method and field objects in any combination. The field must be public. This field is not valid for edge-cacheable components. For example:

  ```
  <component id="getUser" type="method"><method>getUserInfo
  <field>name</field></method></component>
  ```

  This method is equivalent to the getUser().getUserInfo().name method.
- Use the required element to specify whether or not this component must return a non-null value for this cache ID to represent a valid cache. If set to `true`, this component must return a non-null value for this cache ID to represent a valid cache ID. If set to `false`, the default, a non-null value is used in the formation of the cache ID and a null value means that this component is not used at all in the ID formation. For example:

  ```
  <required>true</required>
  ```
- Use the value element to specify values that must match to use this component in cache ID formation. For example:

  ```
  <component id="getUser" type="method"><value>blue</value>
  <value>red</value> </component>
  ```
- Use the not-value element to specify values that must not match to use this component in cache ID formation. This method is similar to value element, but instead prescribes the defined values from caching. You can use multiple not-value elements when more than one value that is not valid exists. For example:

  ```
  <component id="getUser" type="method">
  <required>true</required>
  <not-value>blue</not-value>
  <not-value>red</not-value></component>
  ```

The component subelement can have either a method and a field element, a value element, or a not-value element. The method and field elements apply to commands only. The following example illustrates the attributes of a component sub-element:

```
<component id="isValid" type="method" ignore-value="true"><component>
```

timeout subelement

The timeout subelement is used to specify an absolute time-to-live (TTL) value for the cache entry. For example,

```
<timeout>value</timeout>
```

where *value* is the amount of time, in seconds, to keep the cache entry. Cache entries that are in memory are kept indefinitely, as long as the entries remain in memory. Cache entries that are stored on disk are evicted if they are not accessed for 24 hours.

inactivity subelement

The inactivity subelement is used to specify a time-to-live (TTL) value for the cache entry based on the last time that the cache entry was accessed. It is a subelement of the cache-id element.

```
<inactivity>value</inactivity>
```

where *value* is the amount of time, in seconds, to keep the cache entry in the cache after the last cache hit.

priority subelement

Use the priority subelement to specify the priority of a cache entry in a cache. The priority weighting is used by the least recently used (LRU) algorithm of the cache to decide which entries to remove from the cache if the cache runs out of storage space. For example,

```
<priority>value</priority>
```

where *value* is a positive integer between 1 and 255 inclusive.

Samples

The following sample keeps the cache entry in the cache for a minimum of 35 seconds and a maximum of 180 seconds. If the cache entry is accessed within each 35 second inactivity period, the inactivity period is extended for another 35 seconds. However, because the timeout element is also configured, the cache entry is always invalidated after 180 seconds. If the cache entry is not accessed within the 35 second period, the entry is removed from the cache.

```
<cache-id>
 <component id="timeout" type="parameter">
  <required>true</required>
 </component>
 <timeout>180</timeout>
 <inactivity>35</inactivity>
 <priority>1</priority>
</cache-id>
```

The following sample keeps the cache entry in the cache for a minimum of 600 seconds. If the cache entry is accessed within each 600 second period, the inactivity period is extended for another 600 seconds. If the cache entry is not accessed within the 600 second period, the cache entry is removed from the cache.

```
<cache-id>
 <component id="timeout" type="parameter">
  <required>true</required>
 </component>
 <inactivity>600</inactivity>
 <priority>1</priority>
</cache-id>
```

In the following sample, the value for inactivity has no meaning because the timeout period is less than the inactivity period. The cache entry is always invalidated after 180 seconds, no matter how often the cache entry is accessed.

```
<cache-id>
 <component id="timeout" type="parameter">
  <required>true</required>
 </component>
 <timeout>180</timeout>
 <inactivity>600</inactivity>
 <priority>1</priority>
</cache-id>
```

property subelement

Use the property subelement to specify generic properties for the cache entry. For example,

```
<property name="key">value</property>
```

where *key* is the name of the property to define, and *value* is the corresponding value.

For example:

```
<property name="description">The Snoop Servlet</property>
```

Property	Valid classes	Meaning
sharing-policy/timeout/priority	All	Overrides the settings for the containing cache entry when the request matches this cache ID.
EdgeCacheable	Servlet	Overrides the settings for the containing cache entry when the request matches this cache ID.

idgenerator and metadatagenerator sub-elements

Use the idgenerator element to specify the class name that is loaded for the generation of the cache ID. The IdGenerator element must implement the com.ibm.websphere.servlet.cache.IdGenerator interface for a servlet or the com.ibm.websphere.webservices.IdGenerator interface for the Web services client cache. An example of the idgenerator element follows:

```
<idgenerator> class name </idgenerator>
```

Where `class name` is the fully-qualified name of the class to use. Define this generator class in a shared library.

Use the metadatagenerator element inside the cache-id element to specify the class name loaded for the metadata generation. The MetadataGenerator class must implement the com.ibm.websphere.servlet.cache.MetaDataGenerator interface for a servlet or the com.ibm.websphere.cache.webservices.MetaDataGenerator interface for Web services client cache. The MetadataGenerator class defines properties like timeout, inactivity, external caching properties or dependencies. An example of the `metadatagenerator` element follows:

```
<metadatagenerator> classname </metadatagenerator>
```

In this example, class name is the fully-qualified name of the class to use. Define this generator class in a shared library.

dependency-id element

Use the dependency-id element to specify additional cache identifiers that associate multiple cache entries to the same group identifier.

The value of the dependency-id element is generated by concatenating the dependency ID base string with the values that are returned by its component elements. If a required component returns a null value, the entire dependency does not generate and is not used. Validate the dependency IDs explicitly through the dynamic cache API, or use the invalidation element. Multiple dependency ID rules can exist in one cache-entry element. All dependency rules run separately.

invalidation element

To invalidate cached objects, the application server must generate unique invalidation IDs. Build invalidation IDs by writing custom Java code or through rules that are defined in the cache policy of each cache entry. The following example illustrates an invalidation in the cache policy:

```
<invalidation>component* | invalidationgenerator? </invalidation>
```

invalidationgenerator subelement

The invalidationgenerator element is used with the Web Services client cache only. Use the invalidationgenerator element to specify the class name to load for generating invalidation IDs. The

InvalidationGenerator class must implement the com.ibm.websphere.cache.webservices.InvalidationGenerator interface. An example of the invalidationgenerator element follows:

```
<invalidationgenerator>class name</invalidationgenerator>
```

In this example, classname is the fully qualified name of the class that implements the com.ibm.websphere.cache.webservices.InvalidationGenerator interface. Define this generator class in a shared library.

Configuring command caching

Cacheable commands are stored in the cache for reuse with a similar mechanism for servlets and JavaServer Pages (JSP) files.

In this case, however, the unique cache IDs are generated based on methods and fields present in the command as input parameters. For example, a **GetStockQuote** command can have a symbol as its input parameter.

A unique cache ID can generate from the name of the command, plus the value of the symbol.

To use command caching you must:

Create a command.

1. Define an interface. The Command interface specifies the most basic aspects of a command.

   You must define the interface that extends one or more of the interfaces in the command package. The command package consists of three interfaces:
   - TargetableCommand
   - CompensableCommand
   - CacheableCommand

   In practice, most commands implement the TargetableCommand interface, which allows the command to run remotely. The code structure of a command interface for a targetable command follows:

   ```
   ...
   import com.ibm.websphere.command.*;
   public interface MyCommand extends TargetableCommand {
         // Declare application methods here
   }
   ```

2. Provide an implementation class for the interface. Write an interface that extends the CacheableCommandImpl class and implements your command interface. This class contains the code for the methods in your interface, the methods inherited from extended interfaces like the CacheableCommand interface, and the required or abstract methods in the CacheableCommandImpl class.

   You can also override the default implementations of other methods provided in the CacheableCommandImpl class.

Command class

Extend one or more of the three interfaces included in the command package to write a command interface. The base interface for all commands is the Command interface.

The Command interface provides only the client-side interface for generic commands and declares three basic methods:

- **isReadyToCallExecute.** This method is called on the client side before the command runs on server.
- **execute.** This method passes the command to the target and returns any data.
- **reset.** This method reverts any output properties to the values they had before the execute method was called so that you can reuse the object.

The implementation class for your interface must contain implementations for the isReadyToCallExecute and reset methods.

CacheableCommandImpl class

Commands are implemented by extending the class CacheableCommandImpl, which implements the CacheableCommand interface.

The CacheableCommandImpl class is an abstract class that provides implementations for some of the methods in the CacheableCommand interface, for example, setting return values. This class declares additional methods that the application must implement, for example, how to run the command.

The code structure of an implementation class for the CacheableCommand interface follows:

```
...
import com.ibm.websphere.command.*;
public class MyCommandImpl extends CacheableCommandImpl
implements MyCommand {
// Set instance variables here      ...
// Implement methods in the MyCommand interface      ...
// Implement abstract methods in the CacheableCommandImpl class
...
}
```

Example: Caching a command object

Cacheable commands are stored in the cache for reuse with a similar mechanism for servlets and JavaServer Pages (JSP) files.

This example of command caching is a simple stock quote command.

The following is a stock quote command bean. It accepts a ticker as an input parameter and produces a price as its output parameter.

```
public class QuoteCommand extends CacheableCommandImpl
{
    private String ticker;
    private double price;
    // called to validate that command input parameters have been set
    public boolean isReadyToCallExecute() {
      return (ticker!=null);
    }
    // called by a cache-hit to copy output properties to this object
    public void setOutputProperties(TargetableCommand fromCommand) {
        QuoteCommand f = (QuoteCommand)fromCommand;
        this.price = f.price;
    }

   // business logic method called when the stock price must be retrieved
    public void performExecute()throws Exception {...}

    //input parameters for the command
    public void setTicker(String ticker) { this.ticker=ticker;}
    public String getTicker() { return ticker;}

    //output parameters for the command
    public double getPrice()  { return price;};
}
```

To cache the above command object using the stock ticker as the cache key and using a 60 second time-to-live, use the following cache policy:

```
<cache>
 <cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
```

```
  <name>QuoteCommand</name>
  <cache-id>
   <component type="method" id="getTicker">
    <required>true</required>
   </component>
   <priority>3</priority>
   <timeout>60</timeout>
  </cache-id>
 </cache-entry>
</cache>
```

Configuring the Web services client cache

Configuring the Web services client cache can improve the performance of your application server by caching the responses from remote Web services for a specified amount of time.

You should have the dynamic cache service enabled. To enable the dynamic cache service, see “Task overview: Using the dynamic cache service to improve performance” on page 1913. Before attempting to configure the Web services client cache, you should understand how to create basic cache policies. See “Configuring cacheable objects with the cachespec.xml file” on page 1946 for more information.

Enabling the Web services client cache is an option to improve the performance of your system by using the dynamic cache service to save responses from remote Web services for a specified amount of time. For more information about the Web Services client cache, see “Web services client cache” on page 1967.

1. Locate the Web Services Description Language (WSDL) file for the remote service. Portions of the WSDL file contain information that you will use in writing your cache policy. For more information about WSDL files, see “WSDL” on page 443. Following is an example of portions of a WSDL file that contains values that are used for the purpose of demonstration.

```
<definitions targetNamespace="http://TradeSample.com/"
  xmlns:tns="http://TradeSample.com/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <message name="getQuoteRequest">
  <part name="symbol" type="xsd:string"/>
 </message>
.....
.....
<binding name="SoapBinding" type="tns:GetQuote">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getQuote">
   <soap:operation soapAction=""/>
   <input name="getQuoteRequest">
    <soap:body namespace="http://TradeSample.com/"
    use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
   ......
   </operation>
</binding>
<service name="GetQuoteService">
 <port binding="tns:SoapBinding" name="SoapPort">
  <soap:address location="http://TradeSample.com:9080/service/getquote"/>
 </port>
</service>
</definitions>
```

   The highlighted text indicates values that are used in writing your cache policy.

2. Choose how you plan to generate the cache id for your Web services client caching. You can build your cache id rules by using one of four options:
   - By calculating a hash of the SOAPEnvelope

- By using SOAPHeader entries
- By using operation and part parameters
- By using custom Java code to build the cache id from input SOAP message content

Using SOAPHeader entries is the best option if you can include information for building cache keys as part of the SOAP header. This method creates easy to read cache keys and can be built without parsing the SOAP body. Use custom Java code to generate a specific cache id based on the SOAP message. If you cannot include the header information, you can calculate the hash of the SOAPEnvelope for performance or parse the SOAP Body for user-friendly cache keys.

3. Develop your cache policy.

   All Web services client cache policies must have the **class** *JAXRPCClient*. The **name** element in each cache entry is the target endpoint location that is defined in the WSDL file. You can find this address in the WSDL file by finding the `<soap:address location="..."/>` tag located in the **port** element. In the WSDL file for this sample, the address is **http://TradeSample.com:9080/service/getquote**. Develop the rest of your cache policy by using one of the following options:

   - **Calculate a hash of the SOAPEnvelope to identify the request**

     ```
     <cache>
      <cache-entry>
       <class>JAXRPCClient</class>
       <name>http://TradeSample.com:9080/service/getquote</name>
       <cache-id>
        <component id="hash" type="SOAPEnvelope"/>
        <timeout>60</timeout>
       </cache-id>
      </cache-entry>
     </cache>
     ```

     Note the **component** attributes to create a cache id based on a hash calculation of the SOAPEnvelope. The cache id for this sample is generated as `http://TradeSample.com:9080/service/getquote:Hash=xxxHashSoapEnvelope`.

   - **Use the SoapHeader to identify the request**

     ```
     <cache>
      <cache-entry>
       <class>JAXRPCClient</class>
       <name>http://TradeSample.com:9080/service/getquote</name>
       <cache-id>
        <component id="urn:stock:getQuote" type="SOAPHeaderEntry"/>
       </cache-id>
      </cache-entry>
     </cache>
     ```

     This cache id is built by using special information in the SOAP header to identify requests for entries in the cache. Specify the **type** as `SOAPHeaderEntry` and the **id** as the `operation` name located in the **binding** element in the WSDL file. The cache id for this sample is generated as `http://TradeSample.com:9080/service/getquote:urn:stock:getQuote=IBM`.

     An example of a SOAP request generated by the client using SOAP Header:

     Note that the **soapenv:actor** attribute must contain `com.ibm.websphere.cache`.

     ```
     POST /wsgwsoap1/soaprpcrouther HTTP/1.1
     SOAPAction: ""
     Context-Type: text/xml; charset=utf-8
     User-Agent: Java/1.4.1
     Host: localhost
     Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
     Connection: keep-alive
     Content-Length: 645

     <?xml version="1.0" encoding="UTF-8"?>
     <soapenv:Envelope
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
     ```

```
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
 <getQuote soapenv:actor="com.ibm.websphere.cache" xmlns="urn:stock">IBM</getQuote>
</soapenv:Header>
<soapenv:Body
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
 <getQuote xmlns="urn:ibmwsgw#GetQuoteSample">
  <symbol xsi:type="xsd:string">IBM</symbol>
 </getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

- **Use operation and part to identify the request**

```
<cache>
 <cache-entry>
  <class>JAXRPCClient</class>
  <name>http://TradeSample.com:9080/service/getquote</name>
  <cache-id>
   <component id="" type="operation">
    <value>http://TradeSample.com/:getQuote</value>
   </component>
   <component id="symbol" type="part"/>
  </cache-id>
 </cache-entry>
</cache>
```

This example uses operation and request parameters. The `operation` can be a method name in the WSDL file located in the **binding** element or a method name in the Document/Literal Invocation (DII). If the namespace of the operation is defined, the value should be formatted as `namespaceOfOperation:nameOfOperation`. The `part` type can be defined in the **message** element of the WSDL file, as a request parameter, or as a request parameter of the DII invocation. Its `id` attribute is the part or parameter name, and the `value` is the part or parameter value. The cache id generated from using operation and request parameters is `http://TradeSample.com:9080/service/getquote:operation=http://TradeSample.com/:getQuote/symbol=IBM`.

An example of the SOAP request generated by the client using operation and part:

```
POST /wsgwsoap1/soaprpcrouter HTTP/1.1
SOAPAction:""
Content-Type: text/xml/charset=utf-8
User-Agent: Java/1.4.1
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Current-Length: 645

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns: soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body
 soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <getQuote xmlns="urn:ibmwsgw#GetQuoteSample">
  <symbol xsi:type="xsd:string">IBM</symbol>
 </getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

- **Use custom Java code to build the cache id from input SOAP message content**

If you use custom Java code to build the cache id, create an ID generator Java class that implements the IdGenerator interface defined in the com.ibm.websphere.cache.webservices.IdGenerator package and add a reference to the class you create in the cachespec.xml file by using the **idgenerator** tag.

You can also implement the com.ibm.websphere.cache.webservices.MetaDataGenerator package to assign cache metadata such as timeout, priority, and dependency ids to cache entries using the **metadatagenerator** tag.

Implement the com.ibm.websphere.cache.webservices.InvalidationGenerator interface and use the **invalidationgenerator** tag in the cachespec.xml file to generate cache ids and to invalidate entries in the cache. The id generated by the invalidation generator can be a cache id or a dependency id.

For example, if you develop an ID generator class named SampleIdGeneratorImpl, a metadata generator class named SampleMetaDataGeneratorImpl, and an invalidation generator class named SampleInvalidationGeneratorImpl, your cachespec.xml file might contain the following:

```
<cache-entry>
 <class>JAXRPCClient</class>
 <name>http://TradeSample.com:9080/service/getquote</name>
 <cache-id>
  <idgenerator>com.mycompany.SampleIdGeneratorImpl</idgenerator>
  <metadatagenerator>
       com.mycompany.SampleMetaDataAndInvalidationGeneratorImpl
    </metadatagenerator>
  <timeout>60</timeout>
 </cache-id>
 <invalidation>http://TradeSample.com:9080/service/GetQuote
  <invalidationgenerator>
       com.mycompany.SampleMetaDataAndInvalidationGeneratorImpl
    </invalidationgenerator>
 </invalidation>
</cache-entry>
```

The SampleIdGeneratorImpl class is a custom Java class that implements the com.websphere.cache.webservices.IdGenerator interface. The SampleIdGeneratorImpl class contains the getID method:

```
String getId(javax.xml.rpc.handler.soap.SOAPMessageContext messageContext)
```

The following is an example of the SampleIdGeneratorImpl.java class.

```
public class SampleIdGeneratorImpl implements IdGenerator {
//The SampleIdGenerator class builds cache keys using SOAP header entries
    public String getId(javax.xml.rpc.handler.soap.SOAPMessageContext
   messageContext) {
   ....
   // retrieve SOAP header entries from SOAPMessage
   SOAPHeader sh = soapEnvelope.getHeader();
     if (sh != null) {
    Iterator it = sh.examineHeaderElements("com.mycompany.actor");
    while (it.hasNext()) {
       SOAPHeaderElement element =
                  (SOAPHeaderElement)it.next();
     Name name = element.getElementName();
     String headerEntryName = name.getLocalName();
     if (headerEntryName.equals("getQuote")){
      String sNamespace = element.getNamespaceURI("");
      if (sNamespace != null && !sNamespace.equals("")) {
          headerEntryName = sNamespace + ":" + headerEntryName;
       String quotes = element.getValue();
      }
      ...
      ...
      // create a method "parseAndSort" to parse and sort quotes
      // By parsing and sorting quotes, you avoid duplicate cache
      // entries.
      // quotes e.g. IBM,CSCO,MSFT,INTC
      // to return a cache key "urn:stock:getQuote=CSCO,IBM,INTC,MSFT"
      String sortQuotes = parseAndSort(quotes);
      cacheKey = headerEntryName + "=" + sortQuotes;
       }
```

```
    }
  return cacheKey;
  }
 }
```

The cache id for this sample is generated as `http://TradeSample.com:9080/service/getquote:urn:stock:symbol=CSCO,IBM,INTC,MSFT`.

The SampleMetaDataAndInvalidationGeneratorImpl class is a custom Java class that implements the `com.websphere.cache.webservices.MetaDataGenerator` interface and the `com.websphere.cache.webservices.InvalidationGenerator` interface. The SampleMetaDataAndInvalidationGeneratorImpl class contains the setMetaData method and the getInvalidationIds method. You can also set up two smaller classes instead of this one large class. For example, create one class for the metadata generator and a different class for the invalidation generator. The following are method prototypes for the setMetaData method and the getInvalidationIds method:

```
void setMetaData (javax.xml.rpc.handler.soap.SOAPMessageContext messageContext,
 com.ibm.websphere.cache.webservices.JAXRPCEntryInfo entryInfo)
String[] getInvalidationIds (javax.xml.rpc.handler.soap.SOAPMessageContext messageContext)
```

An example of the SampleMetaDataAndInvalidationGeneratorImpl.java class follows:

```
public class SampleMetaDataAndInvalidationGeneratorImpl implements
  MetaDataGenerator, InvalidationGenerator {
    //assigns time limit, and priority metadata
    public void setMetadata(javax.xml.rpc.handler.soap.SOAPMessageContext messageContext,
  com.ibm.websphere.cache.webservices.JAXRPCEntryInfo entryInfo) {
   ....

  // retrieve SOAP header entries from SOAPMessage
   SOAPHeader sh = soapEnvelope.getHeader();
     if (sh != null) {
    Iterator it = sh.examineHeaderElements("com.mycompany.actor");
    while (it.hasNext()) {
       SOAPHeaderElement element =
                 (SOAPHeaderElement)it.next();
     Name name = element.getElementName();
     String headerEntryName = name.getLocalName();
                                if (headerEntryName.equals("metadata")) {
     // retrieve each metadata element and set metadata
                 entryInfo.setTimeLimit(timeLimit);
                entryInfo.setPriority(priority);
            }
        }
    }

    //builds invalidation ids using SOAP header.
    public String[] getInvalidationIds(javax.xml.rpc.handler.soap.SOAPMessageContext
      messageContext) {   ....
   // retrieve SOAP header entries from SOAPMessage
                String[] invalidationIds = new String[1];
   SOAPHeader sh = soapEnvelope.getHeader();
     if (sh != null) {
    Iterator it = sh.examineHeaderElements("com.mycompany.actor");
    while (it.hasNext()) {
       SOAPHeaderElement element =
                 (SOAPHeaderElement)it.next();
     Name name = element.getElementName();
     String headerEntryName = name.getLocalName();
                                if (headerEntryName.equals("invalidation")) {
     String sNamespace = element.getNamespaceURI("");
     if (sNamespace != null && !sNamespace.equals("")) {
         headerEntryName = sNamespace + ":symbol";
      String quotes = element.getValue();
     }
     ...
     ...
```

```
     // create a method "parseAndSort" to parse and sort quotes
     // By parsing and sorting quotes, you avoid duplicate cache
     // entries.
     // quotes e.g. SUNW,NT
     // to return a cache key "urn:stock:symbol=NT,SUNW"
     String sortQuotes = parseAndSort(quotes);
     invalidationIds[0] = headerEntryName + "=" sortQuotes;
             }
     }
  return invalidationIds;
    }
}
```

The invalidation id for this sample is generated as:

```
http://TradeSample.com:9080/service/getquote:urn:stock:symbol=NT,SUNW
```

An example of the SOAP request generated by the client when using custom Java code follows:

```
POST /wsgwsoap1/soaprpcrouter HTTP/1.1
SOAPAction: ""
Context-type: text/xml, charset=utf-8
User-Agent: Java/1.4.1
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length:645

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
 <getQuote soapenv:actor="com.mycompany.actor"
   xmlns="urn:stock">IBM,CSCO,MSFT,INTC</getQuote>
  <metaData soapenv:actor="com.mycompany.actor" xmlns="urn:stock">
   <priority>10</priority>
   <timeLimit>30000</timeLimit>
  </metaData>
  <invalidation soapenv:actor="com.mycompany.actor"
   xmlns="urn:stock">SUNW, NT</invalidation>
</soapenv:Header>
<soapenv:Body
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
<getQuote xmlns="urn:ibmwsgw#GetQuoteSample">
 <symbol xsi:type="xsd:string">IBM,CSCO,MSFT,INTC</symbol>
</getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

4. Save the cache policy to the appropriate directory.
   - If you are using the Web Services Gateway on SOAP channel 1, the directory is: <app_server_root>\installedApps\wsgwsoap1.*servername*.*nodename*.ear/wsgwsoap.war/WEB-INF
   - If you are using a simple JAX-RPC client in your application to invoke remote Web services, save your cache policy in the Web module WEB-INF of your JAX-RPC application.

You can monitor the results of your Web services client cache policy by using the dynamic cache monitor. See “Displaying cache information” on page 1977 for more information.

Web services client cache

The Web services client cache is a part of the dynamic cache service that is used to increase the performance of Web services clients by caching responses from remote Web services.

After a response is returned from a remote Web service, the response is saved in the client cache on the application server. Any identical requests that are made to the same remote Web service are then

responded to from the cache for a specified period of time. The Web services client cache relies primarily on time-based invalidations because the target web service can be outside of your enterprise network and unaware of your client caching. Therefore, you can specify the amount of time in the cache and the rules to build cache entry IDs in the cache in your client application.

The Web services client cache is provided as a JAX-RPC handler on your application server. This JAX-RPC cache handler intercepts the SOAP requests that flow through it from application clients. It then identifies a cache policy based on the target Web service. After a policy is found, all the cache ID rules are evaluated one by one until a valid rule is detected.

You can build cache id rules for Web services in three ways:

- By calculating a hash of the SOAPEnvelope
- By using SOAP header entries
- By using operation and part parameters
- By using custom Java code to build the cache id from an input SOAP message

Building the cache id rules using the SOAP header is faster because it does not have to parse the entire body of the SOAP document. For more information about building the cache policies for the Web services client cache, see “Configuring the Web services client cache” on page 1962.

Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache

By using the DistributedMap or DistributedObjectCache interfaces, Java 2 platform, Enterprise Edition (J2EE) applications and system components can cache and share Java objects by storing a reference to the object in the cache.

Enable the dynamic cache service. See “Enabling the dynamic cache service” on page 1929 for more information.

The DistributedMap and DistributedObjectCache interfaces are simple interfaces for the dynamic cache. Using these interfaces, J2EE applications and system components can cache and share Java objects by storing a reference to the object in the cache. The default dynamic cache instance is created if the dynamic cache service is enabled in the administrative console. This default instance is bound to the global Java Naming and Directory Interface (JNDI) namespace using the name services/cache/distributedmap.

Multiple instances of the DistributedMap and DistributedObjectCache interfaces on the same Java virtual machine (JVM) enable applications to separately configure cache instances as needed. Each instance of the DistributedMap interface has its own properties that can be set using “Object cache instance settings” on page 1971.

Tip: For more information about the DistributedMap and DistributedObjectCache interfaces, see the API documentation for the com.ibm.websphere.cache package. See Reference: Generated API documentation for more information.

Important: If you are using custom object keys, you must place your classes in a shared library. You can define the shared library at cell, node, or server level. Then, in each server create a class loader and associate it with the shared library that you defined. See the *Setting up the application serving environment* PDF for more information.

There are three methods for configuring and using cache instances.

- **Method 1 - Administrative console** You can create additional cache instances using the administrative console.

1. In the administrative console, select **Resources > Object cache instances** and create a new object cache instance.

If you defined two object cache instances in the administrative console with JNDI names of **services/cache/instance_one** and **services/cache/instance_two**, you can use the following code to look up the cache instances:

```
 InitialContext ic = new InitialContext();
DistributedMap dm1 = (DistributedMap)ic.lookup("services/cache/instance_one");

DistributedMap dm2 = (DistributedMap)ic.lookup("services/cache/instance_two");

// or

InitialContext ic = new InitialContext();
DistributedObjectCache dm1 = (DistributedObjectCache)ic.lookup("services/cache/instance_one");

DistributedObjectCache dm2 = (DistributedObjectCache)ic.lookup("services/cache/instance_two");
```

- **Method 2 - Properties file** You can create cache instances using the `cacheinstances.properties` file and package the file in your Enterprise Archive (EAR) file.

  Following is an example of how you can create additional cache instances using the `cacheinstances.properties` file:

```
cache.instance.0=/services/cache/instance_one

cache.instance.0.cacheSize=1000

cache.instance.0.enableDiskOffload=true

cache.instance.0.diskOffloadLocation=${app_server_root}/diskOffload

cache.instance.0.flushToDiskOnStop=true

cache.instance.0.useListenerContext=true

cache.instance.0.enableCacheReplication=false

cache.instance.0.disableDependencyId=false

cache.instance.0.htodCleanupFrequency=60

cache.instance.1=/services/cache/instance_two

cache.instance.1.cacheSize=1500

cache.instance.1.enableDiskOffload=false

cache.instance.1.flushToDiskOnStop=false

cache.instance.1.useListenerContext=false

cache.instance.1.enableCacheReplication=true

cache.instance.1.replicationDomain=DynaCacheCluster

cache.instance.1.disableDependencyId=true
```

  The preceding example creates two cache instances named `instance_one` and `instance_two`. `instance_one` has a cache entry size of 1,000 and `instance_two` has a cache entry size of 1,500. Disk offload is enabled in `instance_one` and disabled in `instance_two`. Use listener context is enabled in `instance_one` and disabled in `instance_two`. Flush to disk on stop is enabled in `instance_one` and disabled in `instance_two`. Cache replication is enabled in `instance_two` and disabled in `instance_one`. The name of the data replication domain for `instance_two` is DynaCacheCluster. Dependency ID support is disabled in `instance_two`.

You must place the cacheinstances.properties file in either your application server or application class path. For example, you can use your application WAR file, WEB-INF\classes directory, or was_root\classes directory. The first entry in the properties file (cache.instance.0) specifies the JNDI name for the cache instance in the global namespace. You can use the following code to look up the cache instance:

```
InitialContext ic = new InitialContext();
DistributedMap dm1 = (DistributedMap)ic.lookup("services/cache/instance_one");
DistributedMap dm2 = (DistributedMap)ic.lookup("services/cache/instance_two");
```

For more information about the DistributedMap and DistributedObjectCache interfaces, see the API documentation for the com.ibm.websphere.cache package.

- **Method 3 - Resource references**

  Note: Method three is an extension to method one or method two, listed above. First use either method one or method two.

  Define a resource-ref in your module deployment descriptor (web.xml and ibm-web-bnd.xmi files) and look up the cache using the java:comp namespace.

Resource-ref example:
File: web.xml

```
<resource-ref id="ResourceRef_1">
  <res-ref-name>dmap/LayoutCache</res-ref-name>
  <res-type>com.ibm.websphere.cache.DistributedMap</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
<resource-ref id="ResourceRef_2">
  <res-ref-name>dmap/UserCache</res-ref-name>
  <res-type>com.ibm.websphere.cache.DistributedMap</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

File: ibm-web-bnd.xmi

```
<?xml version="1.0" encoding="UTF-8"?>
<webappbnd:WebAppBinding xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:webappbnd="webappbnd.xmi"
xmlns:webapplication="webapplication.xmi" xmlns:commonbnd="commonbnd.xmi"
xmlns:common="common.xmi"
xmi:id="WebApp_ID_Bnd" virtualHostName="default_host">
  <webapp href="WEB-INF/web.xml#WebApp_ID"/>
  <resRefBindings xmi:id="ResourceRefBinding_1" jndiName="services/cache/instance_one">
    <bindingResourceRef href="WEB-INF/web.xml#ResourceRef_1"/>
  </resRefBindings>
  <resRefBindings xmi:id="ResourceRefBinding_2" jndiName="services/cache/instance_two">
    <bindingResourceRef href="WEB-INF/web.xml#ResourceRef_2"/>
  </resRefBindings>
</webappbnd:WebAppBinding>
```

The following example shows how to look up the resource-ref:

```
InitialContext ic = new InitialContext();
  DistributedMap dm1a =(DistributedMap)ic.lookup("java:comp/env/dmap/LayoutCache");
  DistributedMap dm2a =(DistributedMap)ic.lookup("java:comp/env/dmap/UserCache");
// or
  DistributedObjectCache dm1a =(DistributedObjectCache)ic.lookup("java:comp/env/dmap/LayoutCache");
  DistributedObjectCache dm2a =(DistributedObjectCache)ic.lookup("java:comp/env/dmap/UserCache");
```

The previous resource-ref example maps java:comp/env/dmap/LayoutCache to /services/cache/instance_one and java:comp/env/dmap/UserCache to /services/cache/instance_two. In the examples, DistributedMap dm1 and dm1a are the same object. DistributedMap dm2 and dm2a are the same object.

Restriction: DistributedMap and DistributedObjectCache do not have authorization or access control associated with the cache entries.

Object cache instance settings

An object cache instance is a location, in addition to the default shared dynamic cache, where any Java 2 Platform, Enterprise Edition (J2EE) application can store, distribute, and share data. This gives applications greater flexibility and better tuning of the cache resources. Use the DistributedMap programming interface to access this cache instance. See the API documentation for more information.

To view this administrative console page, click **Resources > Cache instances > Object cache instances > *cache_instance_name***.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Use this name when looking up a reference to this cache instance. The results return a `DistributedMap` object.

Description:

Specifies a description for the resource. This field is optional.

Category:

Specifies a category string to classify or group the resource. This field is optional.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands.

Default	2000
Range	100 - 200,000

Default priority:

Specifies the default priority for servlets that can be cached. This value determines how long an entry stays in a full cache.

The recommended value is one. The range is one through 255.

Enable disk offload:

Specifies if disk offloading is enabled.

If you have disk offload disabled, when a new entry is created while the cache is full, the priorities are configured for each entry and the least recently used algorithm are used to remove the entry from the cache in memory. If you enable disk offload, the entry that would be removed from the cache is copied to the local file system. The location of the file is specified by the disk offload location.

Default	false

Offload location:

Specifies the directory that is used for disk offload.

If disk offload location is not specified, the default location, `${WAS_TEMP_DIR}/node/server name/_dynacache/cache JNDI name` will be used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, `${USER_INSTALL_ROOT}/diskoffload` generates the location as `${USER_INSTALL_ROOT}/diskoffload/node/server name/cache JNDI name`. This value is ignored if disk offload is not enabled.

The default value of the ${WAS_TEMP_DIR} property is `${USER_INSTALL_ROOT}/temp`. If you change the value of the ${WAS_TEMP_DIR} property after starting WebSphere Application Server, but do not move the disk cache contents to the new location:

- The Application Server creates a new disk cache file at the new disk offload location.
- If the Flush to disk setting is enabled, all the disk cache content at the old location is lost when you restart the Application Server

Flush to disk:

Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if `Enable Disk Offload` is not selected.

Default	false

Limit disk cache size in GB:

Specifies a value for the maximum disk cache size in GB. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache size in entries:

Specifies a value for the maximum disk cache size in number of entries. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache entry size:

Specifies a value for the maximum size of an individual cache entry in MB. Any cache entry larger than this, when evicted from memory, will not be offloaded to disk. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Performance settings:

Specifies the level of performance that is required by the disk cache. This setting applies only if **enableDiskOffload** is specified for the cache. Performance levels determine how memory resources

should be used on background activity such as cache cleanup, expiration, garbage collection, and so on. This setting applies only if enable disk offload is specified for the cache.

High performance and high memory usage	Indicates that all metadata will be kept in memory.
Balanced performance and balanced memory usage	Indicates some metadata will be kept in memory. This is the default performance setting and will provide an optimal balance of performance and memory usage for most users.
Low performance and low memory usage	Indicates that limited metadata will be kept in memory.
Custom performance	Indicates that the administrator will explicitly configure the memory settings that will be used to support the above background activity. The administrator sets these values using the **DiskCacheCustomPerformanceSettings** object.

Disk cache cleanup frequency:

Specifies a value for the disk cache cleanup frequency, in minutes. If this value is set to 0, the cleanup runs only at midnight. This setting applies only when the Disk Offload Performance Level is low, balanced, or custom. The high performance level does not require disk cleanup, and this value is ignored.

Value	0 to 1440

Maximum buffer for cache identifiers per metaentry:

Specifies a value for the maximum number of cache identifiers that are stored for an individual dependency ID or template in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk offload performance level is custom.

Value	100 to MAXINT

Maximum buffer for dependency identifiers:

Specifies a value for the maximum number of dependency identifier buckets in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	100 to MAXINT

Maximum buffer for templates:

Specifies a value for the maximum number of template buckets that are in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	10 to MAXINT

Eviction policy algorithm:

Specifies the eviction algorithm that the disk cache will use to evict entries once the high threshold is reached. This setting applies only if enable disk offload is specified for the cache.

None	No eviction policy, so the disk cache can grow until it reaches its limit at which time the dynamic cache service stops writing to disk
Random	When the disk size reaches a high threshold limit, the disk cache garbage collector wakes up and randomly picks entries on the disk and evicts them until the size reaches a low threshold limit.
Size	When the disk size reaches a high threshold limit, the disk cache garbge collector wakes up and picks the largest entries on the disk and evicts them until the disk size reaches a low threshold limit.

High threshold:

Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The disk cache garbage collector is awaken when the disk size exceeds high threshold limit. The lower value limits disk cache size in GB and disk cache size in entries. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Low threshold:

Specifies when the eviction policy ends. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value limits disk cache size in GB and disk cache size in entries. The disk cache garbage collector, when awaken, evicts entries until the disk size reaches the low threshold limit. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Use listener context:

Set this value to true to have invalidation events sent to registered invalidation listeners using the Java 2 Platform, Enterprise Edition (J2EE) context of the listener. If you want to use listener J2EE context for callback, set this value to **true**. If you want to use the caller thread context for callback, set this to **false**.

Dependency ID support:

Specifies that the dynamic cache service, supports cache entry dependency IDs. Disable this option if you do not need to use dependency IDs. Dependency IDs specify additional cache group identifiers that associate multiple cache entries to the same group identifier in your cache policy.

This option might not be available for cache instances that were created with a previous version of WebSphere Application Server.

Default	true

Enable cache replication:

Use cache replication to enable sharing of cache IDs, cache entries, and cache invalidations with other servers in the same replication domain.

This option might be unavailable for cache instances created with a previous version of WebSphere Application Server.

Full group replication domain:

Specifies a replication domain from which your data is replicated.

Specifies a replication domain from which your data is replicated. Choose from any replication domains that have been defined. If there are no replication domains listed, you must create one during cluster creation or manually in the administrative console by clicking **Environment > Internal replication domains > New**. The replication domain you choose to use with the dynamic cache service must be using a Full group replica. Do not share replication domains between replication consumers. Dynamic cache should use a different replication domain from session manager or stateful session beans.

Replication type:

Specifies the global sharing policy for this cache instance.

The following settings are available:

- **Both push and pull** sends the cache ID of newly updated content to other servers in the replication domain. Then, if one of the other servers requests the content, and that server has the ID of the cache entry for the previously updated content, it will retrieve the content from the publishing server. If a request is made for an ID which has not been previously published, the server assumes it does not exist in the cluster and creates a new entry.
- **Pull only** shares cache entries for this object between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server runs the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended.
- **Push only** sends the cache ID and cache content of new content to all other servers in the replication domain.
- The sharing policy of **Not Shared** results in the cache ID and cache content not being shared with other servers in the replication domain.

The default setting for a an environment without clustering is **Not Shared**. When enabling replication, the default value is **Not Shared**.

Push frequency:

Specifies the time, in seconds, to wait before pushing new or modified cache entries to other servers.

A value of `0` (zero) sends the cache entries immediately. Setting this property to a value greater than `0` (zero) results in a ″batch″ push of all cache entries that are created or modified during the time period. The default is `1` (one).

Object cache instance collection

Use this page to configure and manage object cache instances, which in addition to the default shared dynamic cache, can store, distribute, and share data for Java 2 Platform, Enterprise Edition (J2EE) applications. Use cache instances to give applications better flexibility and tuning of the cache resources.

To view this administrative console page, click **Resources > Cache instances > Object cache instances**.

Use the DistributedObjectCache programming interface to access the cache instances. For more information about the DistributedObjectCache application programming interface, see the API documentation.

Scope:

Specify CELL SCOPE to view and configure cache instances that are available to all servers within the cell. Specify NODE SCOPE to view and configure cache instances that are available to all servers with the particular node. Specify SERVER SCOPE to view and configure cache instances that are available only on the specific server.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Use this name when looking up a reference to this cache instance. The results return a `DistributedMap` object.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands. The default is 2000.

The minimum value is 100, with no set maximum value.

Invalidation listeners

Invalidation listener mechanism uses Java events for alerting applications when contents are removed from the cache.

Applications implement the InvalidationListener interface (defined in the `com.ibm.websphere.cache` package) and register it to the cache using the DistributedMap interface. Listeners receive InvalidationEvents (defined in the `com.ibm.websphere.cache` package) when entries from the cache are removed, due to an explicit user invalidation, timeout, least recently used (LRU) eviction, cache clear, or disk timeout. Applications can immediately recalculate the invalidated data and prime the cache before the next user request.

Enable listener support in DistributedMap before registering listeners. DistributedMap can also be configured to use the invalidation listener Java 2 Platform, Enterprise Edition (J2EE) context from registration time during callbacks. Setting the value of the custom property useListenerContext to true enables the invalidation listener J2EE context for callbacks. See Cache instance settings for more information.

The following example shows how to set up an invalidation listener:

```
dmap.enableListener(true);  // Enable cache invalidation listener.
InvalidationListener listener = new MyListenerImpl();  //Create invalidation listener object.
dmap.addInvalidationListener(listener);  //Add invalidation listener.
      :
      :
      :
dmap.removeInvalidationListener(listener);  //Remove the invalidation listener.
//This increases performance.
dmap.enableListener(false);  // Disable cache invalidation listener.
//This increases performance.
```

For more information about invalidation listeners, see Reference: Generated API documentation for the `com.ibm.websphere.cache` package.

Disabling template-based invalidations during JSP reloads

By setting the Java virtual machine (JVM) `com.ibm.ws.cache.CacheConfig.disableTemplateInvalidation` custom property to **true**, the template-based invalidations are disabled during JSP reloads.

To set any of these JVM custom properties, complete the following steps:

1. In the administrative console, click **Servers > Application servers > *server_name* > Process definition > Java virtual machine > Custom properties > New**.
2. Enter `com.ibm.ws.cache.CacheConfig.disableTemplateInvalidation` in the **Name** field.
3. Enter `true` in the **Value** field.
4. Save the property and restart WebSphere Application Server.

Using object cache instances

Perform this task so that your application can access dynamic cache object cache instances with the DistributedMap or DistributedObjectCache interfaces.

Before you begin, enable the dynamic cache service. See “Enabling the dynamic cache service” on page 1929 for more information.

Using object cache instances can improve the performance of your application because you can programmatically store and share frequently used objects. By using object cache instances, you also have the necessary control over the dynamic cache when you are running multiple applications in an application server. See “Cache instances” on page 582 for more information.

1. Configure one or more cache instances.
   a. In the administrative console, click **Resources > Cache instances > Object cache instances**.
   b. Specify the scope for the cache instance.

      Cell scope makes the cache instance available to all servers within the cell. Node scope makes the cache instance available to all servers on the particular node. Cluster scope makes the cache instance available to all members in a specified cluster. Server scope makes the cache instance available to only the selected server. You can mix scopes if necessary.
   c. Click **Apply** after changing the scope.
   d. Click **New**.
   e. Enter the Java Naming and Directory Interface (JNDI) name for this cache instance.

      This is name that you pass to the InitialContext lookup() method from within your application. For example, `services/cache/instance_one`.
   f. Enter or modify other properties as needed.
2. Update your application. To store and retrieve objects in an object cache instance, you need a DistributedMap or DistributedObjectCache reference for the named object cache instance. See “Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache” on page 1968 for more information.

You configured object cache instances that you can access programmatically with the DistributedMap and DistributedObjectCache interfaces.

Displaying cache information

Use this task to monitor the activity of the dynamic cache service.

The dynamic cache monitor is an installable Web application that displays simple cache statistics, cache entries, and cache policy information for servlet cache instances.

1. Use the administrative console to install the cache monitor application from the *app_server_root*`/installableApps` directory. The name of the application is `CacheMonitor.ear`. For more information

about installing applications, see “Installing application files with the console” on page 32. Install the cache monitor onto the application server you are want to monitor.

Installing the cache monitor on the *admin_host* (port 906*x*) is more secure than installing it on the *default_host* (908*x*), and so it is preferable to install it onto the *admin_host*.

2. Configure the Web container transport chain and host alias for the server with cache monitor installed.
   a. If you installed the cache monitor on the *admin_host* (port 906*x*), check if a Web container transport chain has been created. Click **Application servers** > *server_name* > **Web container settings** > **Web container transport chains**. If a Web container transport chain (port 906*x*) does not exist you must create a Web container transport chain in the *admin_host* for this server. If you are using *server1*, a Web container transport chain is installed by default for admin port 9060.
   b. Add a host alias for the port your server is using. Click **Environment** > **Virtual hosts** > *host_type* > **Host aliases** and create a new **Host name** and **Port** to add to the list.
   c. You can then access the cache monitor using `http://your_host_name:your_port_number/cachemonitor`.

   Tip: You can find the port number in the `SystemOut.log` file. Look for message `TCPC0001I` or `SRVE0171I`.
3. Access the cache monitor using a Web browser and the URL `http://your host_name:your port_number/cachemonitor`, where *your port_number* is the port associated with the host on which you installed the cache monitor application.
4. Verify the list of cache instances that are shown. For each cache instance, you can perform the following actions:

   Tip: You must select the servlet cache instance that you want to monitor. If you do not use servlet cache instances by using <cache-instance> tags in your cachespec.xml file, all the content is in the **baseCache** instance.

   - View the Statistics page and verify the cache configuration and cache data. Click **Reset Statistics** to reset the counters.
   - View the Cache Policies page to see which cache policies are currently loaded in the dynamic cache. Click on a template to view the cache ID rules for the template.
   - View the Cache Contents page to examine the contents that are currently cached in memory.
   - View the Edge Statistics page to view data about the current ESI processors configured for caching. Click **Refresh Statistics** to see the latest statistics or content from the ESI processors. Click **Reset Statistics** to reset the counters.
   - View the Disk Offload page to view the disk configuration, statistics, and the content that is currently off-loaded from memory to disk.

   When you are viewing contents on memory or disk, click on a template to view all entries for that template, click on a dependency ID to view all entries for the ID, or click on the cache ID to view all the data that is cached for that entry.
5. Use the cache monitor to perform basic operations on data in a cache instance.

   Remove an entry from cache
   Click **Invalidate** when viewing a cache entry.

   Remove all entries for a certain dependency ID
   Click **Invalidate** when viewing entries for a dependency ID.

   Remove all entries for a certain template
   Click **Invalidate** when viewing entries for a template.

   Move an entry to the front of the Least Recently Used queue to avoid eviction
   Click **Refresh** when viewing a cache entry.

   Move an entry from disk to cache
   Click **Send to Memory** when viewing a cache entry on disk.

   Clear the entire contents of the cache
   Click **Clear Cache** while viewing statistics or contents.

Clear the contents on the ESI processors

Click **Clear Cache** while viewing ESI statistics or contents.

Clear the contents of the disk cache

Click **Clear Disk** while viewing disk contents.

Cache monitor

Cache monitor is an installable Web application that provides a real-time view of the current state of dynamic cache. You use it to help verify that dynamic cache is operating as expected. The only way to manipulate the data in the cache is by using the cache monitor. It provides a GUI interface to manually change data.

Cache monitor provides a way to:

- **Verify the configuration of dynamic cache**

  After you create multiple servlet cache instances in the administrative console, you can configure properties, including the maximum size of the cache and disk offload location on each cache instance, as well as advanced features such as controlling external caches. You can verify the configuration of the dynamic cache by viewing of the configured features and properties in the cache monitor.
- **Verify the cache policies**

  To cache an object, unique IDs must be generated for different invocations of that object. To create unique IDs for each object, provide rules for each cacheable object in the `cachespec.xml` file, found inside the Web module `WEB-INF` or `enterprise bean META-INF` directory. See “Cachespec.xml file” on page 1947 for more information. Each cacheable object can have multiple cache ID rules that run in sequence until either a rule returns a cache ID or no more rules remain. If none of the cache ID generation rules produce a valid cache ID, then the object is not cached. There can be multiple `cachespec.xml` files with multiple cache ID rules. With cache monitor, you can verify the policies of each object. You can also view all of the cache polices for each cache instance that is currently loaded in dynamic cache. This view is also convenient to verify that the `cachespec.xml` file was read by the dynamic cache without errors.
- **Monitor cache statistics**

  You can view the essential cache data, such as number of cache hits, cache misses, and number of entries in each cache instance. With this data, you can tune the cache configuration to improve the dynamic cache performance. For example, if the number of used entries is often high, and entries are being removed and recreated, consider increasing the maximum size of the cache or enabling disk offload.
- **Monitor the data flowing through the cache**

  Once a cacheable object is invoked, dynamic cache creates a cache entry for it that contains the output of the actions that are performed and metadata, such as time to live, sharing policy, and so on. Entries are distinguished by a unique ID string that is based on the rules specified in the `cachespec.xml` file for the particular object name. Objects with the same name might generate multiple cache IDs for different invocations, based on request parameters and attributes for each invocation. You can view of all the cache entries that are in the cache instance, based on the unique ID. You can also view the group of cache entries that share a common name (also known as template). Cache entries can also be grouped together by a dependency ID, which is used to invalidate the entire group of entries dependent on a common entity. Therefore, cache monitor also provides a view of the group of cache entries that share a common dependency ID.

  For each entry, cache monitor also displays metadata, such as time to live, priority and sharing-policy, and provides a view of the output that has been cached. This helps the customer to verify which pages have been cached, that the pages have been cached in the correct cache instance with the right attributes such as time to live, priority, and that the pages have the right content.
- **Monitor the data in the edge cache**

  Dynamic cache provides support to recognize the presence of an Edge Side Include (ESI) processor and to generate ESI include tags and appropriate cache policies for edge cacheable fragments. With the ESI processor, you can cache whole pages, as well as fragments, providing a higher cache hit ratio. There can be multiple ESI processors running on multiple hosts configured for caching.

You can view a list of all ESI processes and their hosts that are enabled for caching. Select a host or a processor, and view its edge cache statistics and the current cache entries.

- **View the data offloaded to the disk**

  By default, when the number of cache entries reaches the configured limit for a given server, cache entries are removed, enabling new entries to enter the cache service. With disk offload, the removed cache entries are copied to disk for future access. You can view the content that is copied to disk that corresponds to the view of the contents cached in memory for each cache instance.
- **Manage the data in the cache**

  You can perform the following basic operations on the data in the cache:
  - Remove an entry from a cache instance
  - Remove all entries for a certain dependency ID
  - Remove all entries for a certain name (template)
  - Move an entry to the front of the least recently used queue to avoid removal of the cache entry
  - Move an entry from the disk to the memory within a cache instance
  - Clear the entire contents of the cache instance
  - Clear the contents of the disk for the cache instance

  With these operations, you can manually change the state of the cache without having to restart the server.

Edge cache statistics:

Cache monitor provides a view of the edge cache statistics.

The following statistics are available:

- **ESI Processors**. The number of processes that are configured as edge caches.
- **Number of Edge Cached Entries**. The number of entries that are currently cached on all edge servers and processes.
- **Cache Hits**. The number of requests that match entries on edge servers.
- **Cache Misses By URL.** A cache policy does not exist on the edge server for the requested template.

  Note:
  - The initial ESI request for a template that has a cache policy on WebSphere Application Server results in a miss.
  - Every request for a template that does not have a cache policy on WebSphere Application Server results in a miss by URL on the edge server.
- **Cache Misses By Cache ID**. The policy for the requested template exists on the edge server, and a cache ID is created, based on the ID rules and the request attributes, but the cache entry for this ID does not exist.

  Note: If the policy exists on the edge server for the requested template, but a cache ID match is not found, based on the ID rules and the request attributes, it is not treated as a cache miss.
- **Cache Timeouts**. The number of entries that are removed from the edge cache based on the timeout value.
- **Evictions**. The number of entries that are removed from the edge cache due to invalidations received from WebSphere Application Server.

Tuning dynamic cache with the cache monitor

Use this task to interpret cache monitor statistics to improve the performance of the dynamic cache service.

Verify that dynamic cache is enabled and that the cache monitor application is installed on your application server.

See the Displaying cache information topic in the *Administering applications and their environment* PDF to configure the cache monitor application.

Use the cache monitor to watch cache hits versus misses. By comparing these two values, you can determine how much dynamic cache is helping your application, and if you can take any additional steps to further improve performance and decrease the cost of processing for your application server.

1. Start cache monitor and click on **Cache Statistics**. You can view the following cache statistics:

Cache statistic	Description
Cache Size	The maximum number of entries that the cache can hold.
Used Entries	The number of cache entries used.
Cache Hits	The number of request responses that are served from the cache.
Cache Misses	The number of request responses that are cacheable but cannot be served from the cache.
LRU Evictions	The number of cache entries removed to make room for new cache entries.
Explicit Removals	The number of cache entries removed or invalidated from the cache based on cache policies or were deleted from the cache through the cache monitor.

2. You can also view the following cache configuration values:

Cache configuration value	Description
Default priority	Specifies the default priority for all cache entries. Lower priority entries are moved from the cache before higher priority entries when the cache is full. You can specify the priority for individual cache entries in the cache policy.
Servlet Caching Enabled	If servlet caching is enabled, results from servlets and JavaServer Pages (JSP) files are cached. See the *Administering applications and their environment* PDF for more information.
Disk Offload Enabled	Specifies if entries that are being removed from the cache are saved to disk. See the *Administering applications and their environment* PDF for more information.

3. Wait for the application server to add data to the cache. You want the number of used cache entries in the cache monitor to be as high as it can go. When the number of used entries is at its highest, the cache can serve responses to as many requests as possible.
4. When the cache has a high number of used entries, reset the statistics. Watch the number of cache hits versus cache misses. If the number of hits is far greater than the number of misses, your cache configuration is optimal. You do not need to take any further actions. If you find a higher number of misses with a lower number of hits, the application server is working hard to generate responses instead of serving the request using a cached value. The application server might be making database queries, or running logic to respond to the requests.
5. If you have a large number of cache misses, increase the number of cache hits by improving the probability that a request can be served from the cache.

   To improve the number of cache hits, you can increase the cache size or configure additional cache policies. See the *Administering applications and their environment* PDF for more information to increase the cache size and to configure cache policies.

By using the cache monitor application, you optimized the performance of the dynamic cache service.

See the *Administering applications and their environment* PDF for more information about the dynamic cache.

Using servlet cache instances

Use this task to configure servlet cache instances.

Before you begin, enable the dynamic cache service. See “Enabling the dynamic cache service” on page 1929 for more information.

Perform this task so that your application can access dynamic cache servlet cache instances. Using servlet cache instances can improve the performance of your application because you can store the output and the side effects of an invoked servlet. Servlet cache instances also give you the necessary control over the cache for multiple applications that are running in an application server. See “Cache instances” on page 582 for more information.

1. Enable servlet caching. See “Configuring servlet caching” on page 1933 for more information.
2. Configure one or more cache instances.
   a. In the administrative console, click **Resources > Cache instances > Servlet cache instances**.
   b. Specify the scope of the cache instance. Specify a scope of cell to make the cache instance available to all the servers that are in the cell. Node scope makes the cache instance available to all servers in a node. Server scope makes the cache instance available to the selected server only. If necessary, you can mix the scopes.
   c. Click **Apply** to save the scope.
   d. Specify the settings for the cache instance. The **Name** and **Java Naming and Directory interface (JNDI)** name fields are required. The **JNDI name** is the name attribute that is specified in the <cache-instance> element in the `cachepec.xml` file. An example of a JNDI name that is specified in the cachespec.xml file follows:

      ```
      <cache-instance name="services/cache/instance_one">
      ```

      In this example, specify `services/cache/instance_one` as the **JNDI name**.
3. Update your application. To use a servlet cache instance, you must specify a <cache-instance> element that has a name that is equal to the JNDI Name for this cache instance.

Servlet cache instance collection

A servlet cache instance is a location, in addition to the default shared dynamic cache, where dynamic cache can store, distribute, and share data. By using servlet cache instances, your applications have greater flexibility and better tuning of the cache resources. The Java Naming and Directory Interface (JNDI) name specified for the cache instance is mapped to the name attribute in the <cache-instance> tag in the `cachespec.xml` configuration file.

To view this administrative console page, click **Resources > Cache instances > Servlet cache instances**.

Scope:

Specify CELL SCOPE to view and configure cache instances that are available to all servers within the cell. Specify NODE SCOPE to view and configure cache instances that are available to all servers with the particular node. Specify SERVER SCOPE to view and configure cache instances that are available only on the specific server.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Specify this name in the name attribute field in the <cache-instance> tag in the `cachespec.xml` configuration file. This tag is used to find the particular cache instance in which to store cache entries.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands. The default is 2000.

The minimum value is 100, with no set maximum value.

Servlet cache instance settings

A servlet cache instance is a location, in addition to the default shared dynamic cache, where dynamic cache can store, distribute, and share data. By using servlet cache instances, your applications have greater flexibility and better tuning of the cache resources. The Java Naming and Directory Interface (JNDI) name specified for the cache instance is mapped to the name attribute in the <cache-instance> tag in the `cachespec.xml` configuration file.

To view this administrative console page, click **Resources > Cache instances > Servlet cache instances > *cache_instance_name***.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Specify this name in the name attribute field in the <cache-instance> tag in the `cachespec.xml` configuration file. This tag is used to find the particular cache instance in which to store cache entries.

Description:

Specifies a description for the resource. This field is optional.

Category:

Specifies a category string to classify or group the resource. This field is optional.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands.

Default	2000
Range	100 - no set maximum value

Default priority:

Specifies the default priority for servlets that can be cached. This value determines how long an entry stays in a full cache.

The recommended value is one.

Enable disk offload:

Specifies if disk offloading is enabled.

If you have disk offload disabled, when a new entry is created while the cache is full, the priorities are configured for each entry and the least recently used algorithm are used to remove the entry from the cache in memory. If you enable disk offload, the entry that would be removed from the cache is copied to the local file system. The location of the file is specified by the disk offload location.

Default	false

Offload location:

Specifies the directory used for disk offload.

If disk offload location is not specified, the default location, `${WAS_TEMP_DIR}/node/server name/_dynacache/cache JNDI name` will be used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, `${USER_INSTALL_ROOT}/diskoffload` generates the location as `${USER_INSTALL_ROOT}/diskoffload/node/server name/cache JNDI name`. This value is ignored if disk offload is not enabled.

The default value of the ${WAS_TEMP_DIR} property is `${USER_INSTALL_ROOT}/temp`. If you change the value of the ${WAS_TEMP_DIR} property after starting WebSphere Application Server, but do not move the disk cache contents to the new location:

- The Application Server creates a new disk cache file at the new disk offload location.
- If the Flush to disk setting is enabled, all the disk cache content at the old location is lost when you restart the Application Server

Flush to disk:

Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if `Enable Disk Offload` is not selected.

Default	false

Limit disk cache size in GB:

Specifies a value for the maximum disk cache size in GB. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache size in entries:

Specifies a value for the maximum disk cache size in number of entries. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache entry size:

Specifies a value for the maximum size of an individual cache entry in MB. Any cache entry larger than this, when evicted from memory, will not be offloaded to disk. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Performance settings:

Specifies the level of performance that is required by the disk cache. This setting applies only if **enableDiskOffload** is specified for the cache. Performance levels determine how memory resources should be used on background activity such as cache cleanup, expiration, garbage collection, and so on. This setting applies only if enable disk offload is specified for the cache.

High performance and high memory usage	Indicates that all metadata will be kept in memory.
Balanced performance and balanced memory usage	Indicates some metadata will be kept in memory. This is the default performance setting and will provide an optimal balance of performance and memory usage for most users.
Low performance and low memory usage	Indicates that limited metadata will be kept in memory.
Custom	Indicates that the administrator will explicitly configure the memory settings that will be used to support the above background activity. The administrator sets these values using the **DiskCacheCustomPerformanceSettings** object.

Disk cache cleanup frequency:

Specifies a value for the disk cache cleanup frequency, in minutes. If this value is set to 0, the cleanup runs only at midnight. This setting applies only when the Disk Offload Performance Level is low, balanced, or custom. The high performance level does not require disk cleanup, and this value is ignored.

Value	0 to 1440

Maximum buffer for cache identifiers per metaentry:

Specifies a value for the maximum number of cache identifiers that are stored for an individual dependency ID or template in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk offload performance level is CUSTOM.

Value	100 to MAXINT

Maximum buffer for dependency identifiers:

Specifies a value for the maximum number of dependency identifier buckets in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	100 to MAXINT

Maximum buffer for templates:

Specifies a value for the maximum number of template buckets that are in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is cusotm.

Value	10 to MAXINT

Eviction policy algorithm:

Specifies the eviction algorithm that the disk cache will use to evict entries once the high threshold is reached. This setting applies only if enable disk offload is specified for the cache.

None	No eviction policy, so the disk cache can grow until it reaches its limit at which time the dynamic cache service stops writing to disk
Random	When the disk size reaches a high threshold limit, the disk cache garbage collector wakes up and randomly picks entries on the disk and evicts them until the size reaches a low threshold limit.
Size	When the disk size reaches a high threshold limit, the disk cache garbge collector wakes up and picks the largest entries on the disk and evicts them until the disk size reaches a low threshold limit.

High threshold:

Specifies when the eviction policy starts. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The disk cache garbage collector is awaken when the disk size exceeds high threshold limit. The lower value limits disk cache size in GB and disk cache size in entries. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Low threshold:

Specifies when the eviction policy ends. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value limits disk cache size in GB and disk cache size in entries. The disk cache garbage collector, when awaken, evicts entries until the disk size reaches the low threshold limit. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Enable cache replication:

Use cache replication to enable sharing of cache IDs, cache entries, and cache invalidations with other servers in the same replication domain.

This option might be unavailable for cache instances created with a previous version of WebSphere Application Server.

Full group replication domain:

Specifies a replication domain from which your data is replicated.

Specifies a replication domain from which your data is replicated. Choose from any replication domains that have been defined. If there are no replication domains listed, you must create one during cluster creation or manually in the administrative console by clicking **Environment > Internal replication domains > New**. The replication domain you choose to use with the dynamic cache service must be using a Full group replica. Do not share replication domains between replication consumers. Dynamic cache should use a different replication domain from session manager or stateful session beans.

Replication type:

Specifies the global sharing policy for this cache instance.

The following settings are available:

- **Both push and pull** sends the cache ID of newly updated content to other servers in the replication domain. Then, if one of the other servers requests the content, and that server has the ID of the cache entry for the previously updated content, it will retrieve the content from the publishing server. If a request is made for an ID which has not been previously published, the server assumes it does not exist in the cluster and creates a new entry.
- **Pull only** shares cache entries for this object between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server runs the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended.
- **Push only** sends the cache ID and cache content of new content to all other servers in the replication domain.
- The sharing policy of **Not Shared** results in the cache ID and cache content not being shared with other servers in the replication domain.

The default setting for a an environment without clustering is **Not Shared**. When enabling replication, the default value is **Not Shared**.

Push frequency:

Specifies the time, in seconds, to wait before pushing new or modified cache entries to other servers.

A value of `0` (zero) sends the cache entries immediately. Setting this property to a value greater than `0` (zero) results in a ″batch″ push of all cache entries that are created or modified during the time period. The default is `1` (one).

Using the DynamicContentProvider interface for dynamic cache

Use this task to configure the DynamicContentProvider interface for cached servlets and JavaServer Pages (JSP) files.

The dynamic cache service should be enabled and you should be using servlet caching. See “Enabling the dynamic cache service” on page 1929 and “Configuring servlet caching” on page 1933 for more information.

A cacheable servlet or JavaServer Pages (JSP) file might contain a state in the response that does not belong to the fragment for that servlet or JSP. When the state changes, the cached servlet or JSP is not valid for caching. Use the com.ibm.websphere.servlet.cache.DynamicContentProvider interface to make the fragment cacheable.

Servlets or JSP files that implement the DynamicContentProvider interface can add user exits in fragments that are cacheable by calling the addDynamicContentProvider(DCP) method on the wrapper response object. When the dynamic cache renders the page, it identifies the user exit and calls the dynamic content provider to add the dynamic content to the rendered page.

1. Provide an implementation class of the com.ibm.websphere.servlet.cache.DynamicContentProvider interface. An example of an implementation follows:

```
class DynamicContentProviderImpl implements com.ibm.websphere.servlet.cache.
   DynamicContentProvider {
    DynamicContentProviderImpl() {}

    public void provideDynamicContent(HttpServletRequest request, OutputStream streamWriter)
           throws IOException {
     String dynamicContent = System.currentTimeMillis();
     streamWriter.write(dynamicContent.getBytes());
    }
    public void provideDynamicContent(HttpServletRequest request, Writer streamWriter)
           throws IOException {
     String dynamicContent = System.currentTimeMillis();
     streamWriter.write(dynamicContent.toCharArray());
    }
   }
```

2. Add user exits to your servlet or JSP file by calling the addDynamicContentProvider(DCP) method on the wrapper response object. An example follows:

```
public class DCPServlet extends CacheTestCase {
    public void performTest(HttpServletRequest request, HttpServletResponse response)
           throws IOException, ServletException {
     out.println("Testing the DCP feature begin "+System.currentTimeMillis());
     DynamicContentProvider dcp = new DynamicContentProviderImpl();
     ServletCacheResponse scr = (ServletCacheResponse)(response);
     scr.addDynamicContentProvider(dcp);
       out.println("Testing the DCP feature end"+System.currentTimeMillis());
    }
   }
```

See “Task overview: Using the dynamic cache service to improve performance” on page 1913 for more information about the other tasks that you can perform with the dynamic cache.

Dynamic query

Using EJB query

The EJB query language is used to specify a query over container-managed entity beans. The language is similar to SQL. An EJB query is independent of the bean’s mapping to a persistent store.

An EJB query can be used in three situations:
- To define a finder method of an EJB entity bean.
- To define a select method of an EJB entity bean.
- To dynamically specify a query using the `executeQuery()` dynamic API.

Finder and select queries are specified in the bean’s deployment descriptor using the `<ejb-ql>` tag; they are compiled into SQL during deployment. Dynamic queries are included within the application code itself.

WebSphere’s EJB query language is compliant with the EJB QL defined in Sun’s EJB 2.1 specification and has additional capabilities as listed in the topic Comparison of EJB 2.x specification and WebSphere Query Language.

- Before using EJB query, familiarize yourself with query language concepts, starting with the topic, EJB Query Language.
- Define an EJB query in one of the following ways:
  - **Application Server Toolkit.** When defining an EJB 2.1 entity bean in an EJB deployment descriptor editor, on the **Beans** page click **Add** under **Queries** and, in the Add Finder Descriptor wizard, define a `find` or `ejbSelect` method. See the online **Application Server Toolkit information** for documentation on wizard options.

- **Rational Application Developer.** When defining an entity bean, specify the <ejb-ql> tag for the finder or select method.
- **Dynamic query service.** Add the executeQuery() method to your application.

See the topic Example: EJB queries.

EJB query language

An EJB query is a string that contains the following elements:
- a SELECT clause that specifies the enterprise beans or values to return;
- a FROM clause that names the bean collections;
- an optional WHERE clause that contains search predicates over the collections;
- an optional GROUP BY and HAVING clause (see Aggregation functions);
- an optional ORDER BY clause that specifies the ordering of the result collection.

Collections of entity beans are identified in EJB queries through the use of their abstract schema name in the query FROM clause.

The elements of EJB query language are discussed in more detail in the following related topics.

Example: EJB queries:

Here is an example EJB schema, followed by a set of example queries:

Table 61. DeptBean schema

Entity bean name (EJB name)	DeptEJB (not used in query)
Abstract schema name	DeptBean
Implementation class	com.acme.hr.deptBean (not used in query)
Persistent attributes (cmp fields)	• deptno - Integer (key) • name - String • budget - BigDecimal
Relationships	• emps - 1:Many with EmpEJB • mgr - Many:1 with EmpEJB

Table 62. EmpBean schema

Entity bean name (EJB name)	EmpEJB (not used in query)
Abstract schema name	EmpBean
Implementation class	com.acme.hr.empBean (not used in query)
Persistent attributes (cmp fields)	• empid - Integer (key) • name - String • salary - BigDecimal • bonus - BigDecimal • hireDate - java.sql.Date • birthDate - java.util.Calendar • address - com.acme.hr.Address
Relationships	• dept - Many:1 with DeptEJB • manages - 1:Many with DeptEJB

Address is a serializable object used as cmp field in EmpBean. The definition of address is as follows:

```
   public class com.acme.hr.Address  extends Object implements Serializable {
public String street;
public String state;
public String city;
```

```
public Integer zip;
     public double distance (String start_location) { ... } ;
     public  String format ( ) { ... } ;
}
```

The following query returns all departments:

```
SELECT OBJECT(d) FROM DeptBean d
```

The following query returns departments whose name begins with the letters ″Web″. Sort the result by name:

```
SELECT OBJECT(d) FROM DeptBean d WHERE  d.name LIKE  'Web%' ORDER BY d.name
```

The keywords SELECT and FROM are shown in uppercase in the examples but are case insensitive. If a name used in a query is a reserved word, the name must be enclosed in double quotes to be used in the query. You can find a list of reserved words in “EJB query: Reserved words” on page 2010. Identifiers enclosed in double quotes are case sensitive. This example shows how to use a cmp field that is a reserved word:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d."select" > 5
```

The following query returns all employees who are managed by Bob. This example shows how to navigate relationships using a path expression:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name='Bob'
```

A query can contain a parameter which refers to the corresponding value of the finder or select method. Query parameters are numbered starting with 1:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name= ?1
```

This query shows navigation of a multivalued relationship and returns all departments that have an employee that earns at least 50000 but not more than 90000:

```
SELECT OBJECT(d) FROM DeptBean d,  IN (d.emps) AS e
WHERE e.salary BETWEEN 50000 and 90000
```

There is a join operation implied in this query between each department object and its related collection of employees. If a department has no employees, the department does not appear in the result. If a department has more than one employee that earns more than 50000, that department appears multiple times in the result.

The following query eliminates the duplicate departments:

```
SELECT DISTINCT OBJECT(d) from DeptBean d,  IN (d.emps) AS e  WHERE e.salary > 50000
```

Find employees whose bonus is more than 40% of their salary:

```
SELECT OBJECT(e) FROM EmpBean e where e.bonus > 0.40 * e.salary
```

Find departments where the sum of salary and bonus of employees in the department exceeds the department budget:

```
SELECT OBJECT(d) FROM DeptBean d where d.budget <
( SELECT SUM(e.salary+e.bonus) FROM IN(d.emps) AS e )
```

A query can contain DB2 style date-time arithmetic expressions if you use java.sql.* datatypes as CMP fields and your datastore is DB2. Find all employees who have worked at least 20 years as of January 1st, 2000:

```
SELECT OBJECT(e) FROM EmpBean e where year(  '2000-01-01' - e.hireDate ) >= 20
```

If the datastore is not DB2 or if you prefer to use java.util.Calendar as the CMP field, then you can use the java millsecond value in queries. The following query finds all employees born before Jan 1, 1990:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate <  631180800232
```

Find departments with no employees:

```
SELECT OBJECT(d) from DeptBean d where d.emps IS EMPTY
```

Find all employees whose earn more than Bob:

```
SELECT OBJECT(e) FROM EmpBean e, EmpBean b
WHERE b.name = 'Bob' AND e.salary + e.bonus > b.salary + b.bonus
```

Find the employee with the largest bonus:

```
SELECT OBJECT(e) from EmpBean e  WHERE e.bonus =
(SELECT MAX(e1.bonus) from EmpBean e1)
```

The above queries all return EJB objects. A finder method query must always return an EJB Object for the home. A select method query can in addition return CMP fields or other EJB Objects not belonging to the home.

The following would be valid select method queries for EmpBean. Return the manager for each department:

```
SELECT  d.mgr FROM DeptBean d
```

Return department 42 manager's name:

```
SELECT  d.mgr.name FROM DeptBean d WHERE  d.deptno = 42
```

Return the names of employees in department 42:

```
SELECT e.name FROM EmpBean e WHERE  e.dept.deptno=42
```

Another way to write the same query is:

```
SELECT e.name from DeptBean d, IN (d.emps) AS e WHERE d.deptno=42
```

Finder and select queries allow only a single CMP field or EJBObject in the SELECT clause. A select query can return aggregate values in Enterprise JavaBeans 2.1 using SUM, MIN, MAX, AVG and COUNT.

```
SELECT max(e.salary) FROM EmpBean e WHERE e.dept.deptno=42
```

The dynamic query API allows multiple expressions in the SELECT clause. The following query would be a valid dynamic query, but not a valid select or finder query:

```
SELECT  e.name, e.salary+e.bonus as total_pay , object(e), e.dept.mgr
FROM  EmpBean e
ORDER BY 2
```

The following dynamic query returns the number of employees in each department:

```
SELECT e.dept.deptno as department_number , count(*) as employee_count
FROM  EmpBean e
GROUP BY  by e.dept.deptno
ORDER BY 1
```

The dynamic query API allows queries that contain bean or value object methods:

```
SELECT object(e), e.address.format( )
FROM EmpBean e EmpBean e
```

FROM clause: The FROM clause specifies the collections of objects to which the query is to be applied. Each collection is specified either by an abstract schema name (ASN) or by a path expression identifying a relationship. An identification variable is defined for each collection.

Conceptually, the semantics of the query is to form a temporary collection of tuples, **R**, with elements consisting of all possible combinations of objects from the collections. This collection is subject to the constraints imposed by any path relationships and by the JOIN operation. The JOIN can be either an *inner* or *outer* join.

The identification variables are bound to elements of the tuple. After forming the temporary collection, the search conditions of the WHERE clause are applied to R, and yield a new temporary collection, **R1**. The ORDER BY, GROUP BY, HAVING, and SELECT clauses are applied to R1 to yield the final result.

```
from_clause::=FROM identification_variable_declaration [, {identification_variable_declaration |
                          collection_member_declaration } ]*

identification_variable_declaration::= range_variable_declaration [join]*

join :=  [ { LEFT [OUTER] | INNER  }]  JOIN  {collection_valued_path_expression | single_valued_path_expression}
 [AS] identifier
```

Examples: Joining collections

`DeptBean` contains records 10, 20, and 30. `EmpBean` contains records 1, 2, and 3 that are related to department 10, and records 4 and 5 that are related to department 20. Department 30 has no employees.

```
SELECT d FROM  DeptBean AS d, EmpBean AS e
WHERE d.name = e.name
```

The comma syntax performs an inner join resulting in all possible combinations. In this example, R would consist of 15 tuples (3 departments x 5 employees). If any collection is empty, then R is also empty. The keyword *AS* is optional.

This example shows that a collection can be joined with itself.

```
SELECT d FROM DeptBean AS d, DeptBean AS d1
```

R would consist of 9 tuples (3 departments x 3 departments).

Examples: Relationship joins

A collection can be a relationship based on a previously declared identifier as in

```
SELECT e FROM DeptBean AS d , IN (d.emps) AS e
```

R would contain 5 tuples. Department 30 would not appear in R because it contains no employees. Department 10 would appear in 3 tuples and department 20 would appear in 2 tuples. IN can only refer to multi-valued relationships. The following is not valid

```
SELECT m FROM EmpBean e,  IN( e.dept.mgr) as m   INVALID
```

When joining with a relationship the alternate syntax INNER JOIN (keyword INNER is optional) can also be used, as shown here.

```
SELECT e FROM DeptBean AS d INNER JOIN d.emps AS e
```

An ASN declaration (**d** in the above query) can be followed by one or more join clauses. The relationship following the JOIN keyword must be related (directly or indirectly) to the ASN declaration. Unlike the case with the IN clause, relationships used in a join clause can be single- or multi-valued. This query has the same semantics as the query

```
SELECT e FROM DeptBean AS d , IN (d.emps) AS e
```

You can use multiple joins together.

```
SELECT m FROM EmpBean e JOIN e.dept d JOIN d.mgr m
```

This is equivalent to

```
SELECT m FROM EmpBean e JOIN e.dept.mgr m
```

Examples: OUTER JOIN

An OUTER JOIN results in a temporary collection that contains combinations of the *left* and *right* operands, subject to the relationship constraints and such that the left operand always appears in R. In the example an outer join results in a temporary collection R that contains department 30, even though the collection **d.emps** is empty. The tuple contains Department 30 along with a NULL value. References to **e** in the query yields a null value.

```
SELECT e FROM DeptBean AS d LEFT OUTER JOIN d.emps AS e
```

The keyword OUTER is optional, as shown here..

```
SELECT e FROM DeptBean AS d LEFT JOIN d.emps AS e
```

You can also use combinations of INNER and OUTER JOIN.

```
SELECT m FROM EmpBean e JOIN e.dept d LEFT JOIN d.mgr m
```

Inheritance in EJB query: If an EJB inheritance hierarchy has been defined for an abstract schema, using the abstract schema name in a query statement implies the collection of objects for that abstract schema as well as all subtypes.

Example: Inheritance

Suppose that bean `ManagerBean` is defined as a subtype of `EmpBean` and `ExecutiveBean` is a subtype of `ManagerBean` in an EJB inheritance hierarchy. The following query returns employees as well as managers and executives:

```
SELECT OBJECT(e) FROM EmpBean e
```

Path expressions: An identification variable followed by the navigation operator (.) and a cmp or relationship name is a path expression.

A path expression that leads to a cmr field can be further navigated if the cmr field is single-valued. If the path expression leads to a multi-valued relationship, then the path expression is terminal and cannot be further navigated. If the path expression leads to a cmp field whose type is a value object, it is possible to navigate to attributes of the value object.

Example: Value object

Assume that `address` is a cmp field for `EmpBean`, which is a value object.

```
SELECT  object(e)  FROM EmpBean e
WHERE  e.address.distance('San Jose') < 10  and e.address.zip = 95037
```

It is best to use the composer pattern to map value object attributes to relational columns if you intend to search on value attributes. If you store value objects in serialized format, then each value object must be retrieved from the database and deserialized. Value object methods can only be done in dynamic queries.

A path expression can also navigate to a bean method. The method must be defined on either the remote or local bean interface. Methods can only be used in dynamic queries. You cannot mix both remote and local methods in a single query statement.

If the query contains remote methods, the dynamic query must be executed using the query remote interface. Using the query remote interface causes the query service to activate beans and create instances of the remote bean interface

Likewise, a query statement with local bean methods must be executed with the query local interface. This causes the query service to activate beans and local interface instances.

Do not use get methods to access cmp and cmr fields of a bean.

If a method has overloaded definitions, the overloaded methods must have different number of parameters.

Methods must have non-void return types and method arguments and return types must be either primitive types byte, short, int, long, float, double, boolean, char or wrapper types from the following list:

Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Date

If any input argument to a method is NULL, it is assumed the method returns a NULL value and the method is not invoked.

A collection valued path expression can be used in the FROM clause as a collection member declaration, and with the IS EMPTY, MEMBER OF, and EXISTS predicates in the WHERE clause.

`FROM EmpBean e WHERE e.dept.mgr.name='Bob'`	OK
`FROM EmpBean e WHERE e.dept.emps.name='BOB'`	INVALID -- cannot navigate through emps because it is multivalued
`FROM EmpBean e, IN (e.dept.emps) e1 WHERE e1.name='BOB'`	OK
`FROM EmpBean e WHERE e.dept.emps IS EMPTY`	OK

WHERE clause: The WHERE clause contains search conditions composed of the following:
- literal values
- input parameters
- expressions
- basic predicates
- quantified predicates
- BETWEEN predicate
- IN predicate
- LIKE predicate
- NULL predicate
- EMPTY collection predicate
- MEMBER OF predicate
- EXISTS predicate
- IS OF TYPE predicate

If the search condition evaluates to TRUE, the tuple is added to the result set.

Literals: A string literal is enclosed in single quotes. A single quote that occurs within a string literal is represented by two single quotes; For example: 'Tom''s'. A string literal cannot exceed the maximum length that is supported by the underlying persistent datastore.

A numeric literal can be any of the following:
- an exact value such as 57, -957, +66
- any value supported by Java long
- a decimal literal such as 57.5, -47.02
- an approximate numeric value such as 7E3, -57.4E-2

A decimal or approximate numeric value must be in the range supported by the underlying persistent datastore.

A boolean literal can be the keyword TRUE or FALSE and is case insensitive.

Input parameters: Input parameters are designated by the question mark followed by a number; For example: `?2`

Input parameters are numbered starting at 1 and correspond to the arguments of the finder or select method; therefore, a query must not contain an input parameter that exceeds the number of input arguments.

An input parameter can be a primitive type of byte, short, int, long, float, double, boolean, char or wrapper types of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Char, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, an EJBObject, or a binary data string in the form of Java byte[].

An input parameter must not have a NULL value. To search for the occurrence of a NULL value the NULL predicate should be used.

Expressions: Conditional expressions can consist of comparison operators and logical operators (AND, OR, NOT).

Arithmetic expressions can be used in comparison expressions and can be composed of arithmetic operations and functions, path expressions that evaluate to a numeric value and numeric literals and numeric input parameters.

String expressions can be used in comparison expressions and can be composed of string functions, path expressions that evaluate to a string value and string literals and string input parameters. A cmp field of type char is handled as if it were a string of length 1.

Binary expressions can be used in comparison expressions and can be composed of path expressions that evaluate to the Java byte[] type as well as input parameters of type byte[].

Boolean expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a boolean value and TRUE and FALSE keywords and boolean input parameters.

Reference expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a cmr field, an identification variable and an input parameter whose type is an EJB reference

Four different expression types are supported for working with date-time types. For portability the java.util.Calendar type should be used. DB2 style date, time and timestamp expressions are supported if the datastore is DB2 and the CMP field is of type java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp. If you use DB2 UDB, you might obtain a syntax error when using the java.sql.Timestamp.ojbect. You must use the syntax TIMESTAMP 'yyyy-mm-dd hh:mm:ss.nnnn'.

A Calendar type can be compared to another Calendar type, an exact numeric literal or input parameter of type long whose value is the standard Java long millisecond value.

The following query finds all employees born before Jan 1, 1990:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate <  631180800232
```

Date expressions can be used in comparison expressions and can be composed of operators + - , date duration expressions and date functions, path expressions that evaluate to a date value, string representation of a date and date input parameters.

Time expressions can be used in comparison expressions and can be composed of operators + - , time duration expressions and time functions, path expressions that evaluate to a time value, string representation of time and time input parameters.

Timestamp expressions can be used in comparison expressions and can be composed of operators + - , timestamp duration expressions and timestamp functions, path expressions that evaluate to a timestamp value, string representation of a timestamp and timestamp input parameters.

Standard bracketing ( ) for ordering expression evaluation is supported.

The operators and their precedence order from highest to lowest are:
- Navigation operator (.)
- Arithmetic operators in precedence order:
  - + - unary
  - * / multiply, divide
  - + - add, subtract
- Comparison operators: =, >, <, >=, <=, <>(not equal)
- Logical operator NOT
- Logical operator AND
- Logical operator OR

Null value semantics: The following describe the semantics of NULL values:
- Comparison or arithmetic operations with an unknown (NULL) value yield an unknown value
- In a Java 2 platform, Enterprise Edition (J2EE) version 1.3 application, a path expression uses an outer-join semantic where a NULL field or cmr value evaluates to NULL. In J2EE version 1.4, the path expression uses an inner-join semantic.
- The IS NULL and IS NOT NULL operators can be applied to path expressions and return TRUE or FALSE. Boolean operators AND, OR and NOT use three valued logic.

AND	**True**	**False**	**Unknown**
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR	**True**	**False**	**Unknown**
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

	NOT
True	False
False	True
Unknown	Unknown

Example: Null value semantics

```
select object(e) from EmpBean where e.salary > 10  and e.dept.budget > 100
```

If salary is NULL the evaluation of `e.salary > 10` returns unknown and the employee object is not returned. If the cmr field dept or budget is NULL evaluation of `e.dept.budget > 100` returns unknown and the employee object is not returned.

```
select object(e) from EmpBean where e.dept.budget is null
```

In J2EE 1.3 if dept or budget is NULL evaluation of `e.dept.budget` is null returns TRUE and the employee object is returned. In J2EE 1.4 the employee object is returned only if budget is NULL.

```
select object(e) from EmpBean e ,  in (e.dept.emps) e1 where e1.salary > 10
```

If dept is NULL, then the multivalued path expression `e.dept.emps` results in an empty collection (not a collection that contains a NULL value). An employee with a null dept value will not be returned.

```
select object(e) from EmpBean e where  e.dept.emps is empty
```

If dept is NULL the evaluation of the predicate in unknown and the employee object is not returned.

```
select object(e) from EmpBean e , EmpBean e1  where e member of  e1.dept.emps
```

If dept is NULL evaluation of the member of predicate returns unknown and the employee is not returned.

Date time arithmetic and comparisons: DATE, TIME and TIMESTAMP values may be compared with another value of the same type. Comparisons are chronological. Date time values can also be incremented, decremented, and subtracted.

If the datastore is DB2, then DB2 string representation of DATE, TIME and TIMESTAMP types can also be used. A string representation of a date or time can use ISO, USA, EUR or JIS format. A string representation of a timestamp uses ISO format.

Format	Date format	Date examples	Time format	Time examples
ISO	yyyy-mm-dd	1987-02-24 1987-2-24	hh.mm.ss	13.50.00 13.50
USA	mm/dd/yyyy	2/24/1987	hh:mm AM or PM	1:50 pm 02:10 AM
EUR	dd.mm.yyyy	24.02.1987 24.2.1987	hh.mm.ss	13.50.00 13.55
JIS	yyyy-mm-dd	1987-02-24	hh:mm:ss	13:50 13:50:05

Example 1: Date time arithmetic comparisons

```
e.hiredate > '1990-02-24'
```

The timestamp of February 24th, 1990 1:50 pm can be represented as follows:

```
'1990-02-24-13.50.00.000000'  or
'1990-02-24-13.50.00'
```

If the datastore is DB2, DB2 decimal durations can be used in expressions and comparisons. A date duration is a decimal(8,0) number that represents the difference between two dates in the format YYYYMMDD. A time duration is a decimal(6,0) number that represents the difference between two time values as HHMMSS. A timestamp duration is a decimal(20,6) number representing the differences between two timestamp values as YYYYMMDDHHMMSS.ZZZZZZ (ZZZZZZ is the number of microseconds and is to the right of the decimal point) .

Two date values (or time values or timestamp values) can be subtracted to yield a duration. If the second operand is greater than the first the duration is a negative decimal number. A duration can be added or subtracted from a datetime value to yield a new datetime value.

Example 2: Date time arithmetic comparisons

`DATE('3/15/2000') - '12/31/1999'` results in a decimal number 215 which is a duration of 0 years, 2 months and 15 days.

Durations are really decimal numbers and can be used in arithmetic expressions and comparisons.

`( DATE('3/15/2000') - '12/31/1999' ) + 14 > 215` evaluates to TRUE.

`DATE('12/31/1999') + DECIMAL(215,8,0)` results in a date value 3/15/2000.

`TIME('11:02:26') - '00:32:56'` results in a decimal number 102930 which is a time duration of 10 hours, 29 minutes and 30 seconds.

`TIME('00:32:56') + DECIMAL(102930,6,0)` results in a time value of 11:02:26.

`TIME('00:00:59') + DECIMAL(240000,6,0)` results in a time value of 00:00:59.

`e.hiredate + DECIMAL(500,8,0) > '2000-10-01'` means compare the hiredate plus 5 months to the date 10/01/2000.

Basic predicates: Basic predicates can be of two forms

```
expression-1  comparison-operator  expression-2
expression-3  comparison-operator ( subselect )
```

The subselect must not return more than one value and the subselect can not return a type of an EJB reference. Boolean types and reference types only support = and <> comparisons.

Example: Basic predicates

```
d.name='Java Development'
e.salary > 20000
e.salary > ( select avg(e.salary) from EmpBean e)
```

Quantified predicates: A quantified predicate compares a value with a set of values produced by a subselect.

```
expression   comparison-operator   SOME | ANY | ALL    ( subselect )
```

The expression must not evaluate to a reference type.

When SOME or ANY is specified the result of the predicate is as follows:
- TRUE if the comparison is true for at least one value returned by the subselect.
- FALSE if the subselect is empty or if the comparison is false for every value returned by the subselect.
- UNKNOWN if the comparison is not true for all of the values returned by the subselect and at least one of the comparisons is unknown because of a null value.

When ALL is specified the result of the predicate is as follows:
- TRUE if the subselect returns empty or if the comparison is true to every value returned by the subselect.
- FALSE if the comparison is false for at least one value returned by the subselect.
- UNKNOWN if the comparison is not false for all values returned by the subselect and at least one comparison is unknown because of a null value.

BETWEEN predicate: The BETWEEN predicate determines whether a given value lies between two other given values.

```
expression  [NOT]  BETWEEN expression-2  AND expression-3
```

The expression must not evaluate to a boolean or reference type.

Example: BETWEEN predicate

```
e.salary BETWEEN 50000 AND 60000
```

is equivalent to:

```
e.salary >= 50000  AND e.salary <= 60000
e.name NOT BETWEEN 'A' AND 'B'
```

is equivalent to:

```
e.name < 'A'  OR e.name > 'B'
```

IN predicate: The IN predicate compares a value to a set of values and can have one of two forms:

```
expression [NOT] IN  ( subselect )
expression [NOT] IN  ( value1, value2,  .... )
```

ValueN can either be a literal value or an input parameter. The expression can not evaluate to a reference type.

Example: IN predicate

```
e.salary IN ( 10000, 15000 )
```

is equivalent to

```
( e.salary = 10000  OR  e.salary = 15000 )
e.salary IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary = ANY   ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
e.salary NOT IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary <> ALL    ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

LIKE predicate: The LIKE predicate searches a string value for a certain pattern.

```
string-expression   [NOT]  LIKE  pattern    [ ESCAPE escape-character ]
```

The pattern value is a string literal or parameter marker of type string in which the underscore (_) stands for any single character and percent (%) stands for any sequence of characters (including empty sequence). Any other character stands for itself. The escape character can be used to search for character _ and %. The escape character can be specified as a string literal or an input parameter.

If the string-expression is null, then the result is unknown.

If both string-expression and pattern are empty, then the result is true.

Example: LIKE predicate

- '' LIKE '' is true
- '' LIKE '%' is true
- e.name LIKE '12%3' is true for '123' '12993' and false for '1234'
- e.name LIKE 's_me' is true for 'some' and 'same', false for 'soome'
- e.name LIKE '/_foo' escape '/' is true for '_foo', false for 'afoo'
- e.name LIKE '//_foo' escape '/' is true for '/afoo' and for '/bfoo'
- e.name LIKE '///_foo' escape '/' is true for '/_foo' but false for '/afoo'

NULL predicate: The NULL predicate tests for null values.

```
single-valued-path-expression IS [NOT] NULL
```

Example: NULL predicate

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

EMPTY collection predicate: To test if a multivalued relationship is empty, use the following syntax:

```
collection-valued-path-expression  IS [NOT]  EMPTY
```

Example: Empty collection predicate

To find all departments with no employees:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d.emps IS EMPTY
```

MEMBER OF predicate: This expression tests whether the object reference specified by the single valued path expression or input parameter is a member of the designated collection. If the collection valued path expression designates an empty collection the value of the MEMBER OF expression is FALSE.

```
{ single-valued-path-expression | input_parameter } [ NOT ] MEMBER [ OF ] collection-valued-path-expression
```

Example: MEMBER OF predicate

Find employees that are not members of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Find employees whose manager is a member of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER  OF d.emps  and d.deptno=?1
```

EXISTS predicate: The exists predicate tests for the presence or absence of a condition specified by a subselect.

```
EXISTS ( subselect )
```

```
EXISTS collection-valued-path-expression
```

The result of EXISTS is true if the subselect returns at least one value or the path expression evaluates to a nonempty collection, otherwise the result is false.

To negate an EXISTS predicate, precede it with the logical operator NOT.

Example: EXISTS predicate

Return departments that have at least one employee earning more than 1000000:

```
SELECT  OBJECT(d) FROM  DeptBean d
WHERE EXISTS ( SELECT  1  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Return departments that have no employees:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT 1 FROM IN (d.emps) e)
```

The above query can also be written as follows:

```
SELECT OBJECT(d) FROM DeptBean d WHERE NOT EXISTS d.emps
```

IS OF TYPE predicate: The IS OF TYPE predicate is used to test the type of an EJB reference. It is similar in function to the Java instance of operator. IS OF TYPE is used when several abstract beans have been grouped into an EJB inheritance hierarchy. The type names specified in the predicate are the bean abstract names. The ONLY option can be used to specify that the reference must be exactly this type and not a subtype.

```
identification-variable IS OF TYPE ( [ONLY] type-1, [ONLY] type-2, ..... )
```

Example: IS OF TYPE predicate

Suppose that bean `ManagerBean` is defined as a subtype of `EmpBean` and `ExecutiveBean` is a subtype of `ManagerBean` in an EJB inheritance hierarchy.

The following query returns employees as well as managers and executives:

```
SELECT  OBJECT(e) FROM EmpBean e
```

If you are interested in objects which are employees and not managers and not executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ONLY EmpBean )
```

If you are interested in object which are managers or executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ManagerBean)
```

The above query is equivalent to the following query:

```
SELECT  OBJECT(e) FROM ManagerBean e
```

If you are interested in managers only and not executives:

```
SELECT  OBJECT(e) FROM  EmpBean e WHERE e IS OF TYPE( ONLY ManagerBean)
```

or:

```
SELECT  OBJECT(e) FROM  ManagerBean e
WHERE  e IS OF TYPE (ONLY ManagerBean)
```

Scalar functions: EJB query contains scalar functions for doing type conversions, string manipulation, and for manipulating date-time values. The list of scalar functions is documented in the topic EJB query: Scalar functions.

Example: Scalar functions

Find employees hired in 1999:

```
SELECT OBJECT(e) FROM EmpBean e where YEAR(e.hireDate) = 1999
```

The only scalar functions that are guaranteed to be portable across backend datastore vendors are the following:

- ABS
- MOD
- SQRT
- CONCAT
- LENGTH
- LOCATE
- SUBSTRING
- UCASE
- LCASE

The other scalar functions should be used only when DB2 is the backend datastore.

EJB query: Scalar functions: EJB query contains scalar built-in functions, as listed below, for doing type conversions, string manipulation, and for manipulating date-time values.

Numeric functions

```
ABS ( < any numeric datatype > ) -> < any numeric datatype >
MOD ( <int>, <int> ) -> int
SQRT ( < any numeric datatype > ) -> Double
```

Type conversion functions

```
CHAR ( < any  numeric datatype > ) ->  string
CHAR ( <  string  > ) ->  string
CHAR ( < any datetime  datatype >  [, Keyword k ]) ->  string
```

Datetime datatype is converted to its string representation in a format specified by the keyword k. The valid keywords values are ISO, USA, EUR or JIS. If k is not specified the default is ISO.

```
BIGINT ( < any numeric datatype > ) -> Long
BIGINT ( < string > ) -> Long
```

The function in the second line of the following code converts the argument to an integer n by truncation, and returns the date that is n-1 days after January 1, 0001:

```
DATE ( < date string > ) -> Date
DATE (  < any numeric datatype>) -> Date
```

The following function returns date portion of a timestamp:

```
DATE( timestamp ) -> Date
DATE ( < timestamp-string > ) -> Date
```

The following function converts number to decimal with optional precision p and scale s.

```
DECIMAL ( < any numeric datatype > [, p [ ,s ] ] ) -> Decimal
```

The following function converts string to decimal with optional precision p and scale s.

```
DECIMAL ( < string > [ , p [ , s ] ] ) -> Decimal
```

```
DOUBLE ( < any numeric datatype > ) -> Double
DOUBLE ( < string > ) -> Double
```

```
FLOAT ( < any numeric datatype > ) -> Double
FLOAT ( < string > ) -> Double
```

Float is a synonym for DOUBLE.

```
INTEGER ( < any numeric datatype > ) -> Integer
INTEGER ( < string > ) -> Integer
```

```
REAL ( < any numeric datatype > ) -> Float
```

```
SMALLINT ( < any numeric datatype ) -> Short
SMALLINT ( < string > ) -> Short
```

```
TIME ( < time > ) -> Time
TIME ( < time-string  > ) -> Time
TIME ( < timestamp > ) -> Time
TIME ( < timestamp-string  > ) -> Time
```

```
TIMESTAMP ( < timestamp > ) -> Timestamp
TIMESTAMP ( < timestamp-string > ) -> Timestamp
```

String functions

```
CONCAT ( <string>, <string>  ) -> String
```

The following function returns a character string representing absolute value of the argument not including its sign or decimal point. For example, `digits( -42.35)` is ″4235″.

```
DIGITS ( Decimal d  ) -> String
```

The following function returns the length of the argument in bytes. If the argument is a numeric or datetime type, it returns the length of internal representation.

```
LENGTH ( < string >  ) -> Integer
```

The following function returns a copy of the argument string where all upper case characters have been converted to lower case.

```
LCASE ( < string > ) -> String
```

The following function returns the starting position of the first occurrence of argument 1 inside argument 2 with optional start position. If not found, it returns 0.

```
LOCATE ( String s1 , String s2  [, Integer start ] ) -> Integer
```

The following function returns a substring of s beginning at character m and containing n characters. If n is omitted, the substring contains the remainder of string s. The result string is padded with blanks if needed to make a string of length n.

```
SUBSTRING ( String s ,  Integer m [ , Integer n ] ) -> String
```

The following function returns a copy of the argument string where all lower case characters have been converted to upper case.

```
UCASE ( < string > ) -> String
```

Date - time functions

The following function returns the day portion of its argument. For a duration, the return value can be -99 to 99.

```
DAY (  Date ) ->  Integer
DAY ( < date-string > ) ->  Integer
DAY ( < date-duration > ) -> Integer
DAY ( Timestamp  ) ->  Integer
DAY ( < timestamp-string > ) ->  Integer
DAY ( < timestamp-duration > ) -> Integer
```

The following function returns one more than number of days from January 1, 0001 to its argument.

```
DAYS ( Date  ) ->  Integer
DAYS ( < Date-string > ) ->  Integer
DAYS ( Timestamp  ) ->  Integer
DAYS ( < timestamp-string > ) ->  Integer
```

The following function returns the hour part of its argument. For a duration, the return value can be -99 to 99.

```
HOUR ( Time ) -> Integer
HOUR ( < time-string > ) -> Integer
HOUR ( < time-duration > ) ->  Integer
HOUR ( Timestamp ) -> Integer
HOUR ( < timestamp-string > ) -> Integer
HOUR ( < timestamp-duration >  ) -> Integer
```

The following function returns the microsecond part of its argument.

```
MICROSECOND ( Timestamp ) -> Integer
MICROSECOND ( < timestamp-string > ) -> Integer
MICROSECOND ( < timestamp-duration >  ) -> Integer
```

The following function returns the minute part of its argument. For a duration, the return value can be -99 to 99.

```
MINUTE ( Time ) -> Integer
MINUTE ( < time-string > ) -> Integer
MINUTE ( < time-duration > ) ->  Integer
MINUTE ( Timestamp ) -> Integer
MINUTE ( < timestamp-string > ) -> Integer
MINUTE ( < timestamp-duration >  ) -> Integer
```

The following function returns the month portion of its argument. For a duration, the return value can be -99 to 99.

```
MONTH (  Date ) ->  Integer
MONTH ( < date-string > ) ->  Integer
MONTH ( < date-duration > ) -> Integer
MONTH ( Timestamp  ) ->  Integer
MONTH ( < timestamp-string > ) ->  Integer
MONTH ( < timestamp-duration > ) -> Integer
```

The following function returns the second part of its argument. For a duration, the return value can be -99 to 99.

```
SECOND ( Time ) -> Integer
SECOND ( < time-string > ) -> Integer
SECOND ( < time-duration > ) ->  Integer
SECOND ( Timestamp ) -> Integer
SECOND ( < timestamp-string > ) -> Integer
SECOND ( < timestamp-duration >  ) -> Integer
```

The following function returns the year portion of its argument. For a duration, the return value can be -9999 to 9999.

```
YEAR (  Date ) ->  Integer
YEAR ( < date-string > ) ->  Integer
YEAR ( < date-duration > ) -> Integer
YEAR ( Timestamp  ) ->  Integer
YEAR ( < timestamp-string > ) ->  Integer
YEAR ( < timestamp-duration > ) -> Integer
```

Aggregation functions: Aggregation functions operate on a set of values to return a single scalar value. You can use these functions in the select and subselect methods. The following example illustrates an aggregation:

```
SELECT  SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

This aggregation computes the total salary for department 20.

The aggregation functions are AVG, COUNT, MAX, MIN, and SUM. The syntax of an aggregation function is illustrated in the following example:

```
aggregation-function  (    [ ALL |  DISTINCT ]  expression )
```

or:

```
COUNT( [ ALL |  DISTINCT ] identification-variable )
```

or:

```
COUNT( * )
```

The DISTINCT option eliminates duplicate values before applying the function. ALL is the default option and does not eliminate duplicates. Null values are ignored in computing the aggregate function except in the cases of COUNT(*) and COUNT(identification-variable), which return a count of all the elements in the set.

If your datastore is Informix, you must limit the expression argument to a single valued path expression when using the COUNT function or the DISTINCT forms of the functions SUM, AVG, MIN, and MAX.

Defining return type

For a select method using an aggregation function, you can define the return type as a primitive type or a wrapper type. The return type must be compatible with the return type from the datastore. The MAX and MIN functions can apply to any numeric, string or datetime datatype and return the corresponding datatype. The SUM and AVG functions take a numeric type as input, and return the same numeric type that is used in the datastore. The COUNT function can take any datatype, and returns an integer.

When applied to an empty set, the SUM, AVG, MAX, and MIN functions can return a null value. The COUNT function returns zero (0) when it is applied to an empty set. Use wrapper types if the return value might be NULL; otherwise, the container displays an ObjectNotFound exception.

Using GROUP BY and HAVING

The set of values that is used for the aggregate function is determined by the collection that results from the FROM and WHERE clause of the query. You can divide the set into groups and apply the aggregation function to each group. To perform this action, use a GROUP BY clause in the query. The GROUP BY clause defines grouping members, which comprise a list of path expressions. Each path expression specifies a field that is a primitive type of byte, short, int, long, float, double, boolean, char, or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp.

The following example illustrates the use of the GROUP BY clause in a query that computes the average salary for each department:

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

In division of a set into groups, a NULL value is considered equal to another NULL value.

Just as the WHERE clause filters tuples (that is, records of the return collection values) from the FROM clause, the groups can be filtered using a HAVING clause that tests group properties involving aggregate functions or grouping members:

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING  COUNT(*) > 3  AND  e.dept.deptno > 5
```

This query returns the average salary for departments that have more than three employees and the department number is greater than five.

It is possible to use a HAVING clause without a GROUP BY clause, in which case the entire set is treated as a single group, to which the HAVING clause is applied.

SELECT clause: For finder and select queries, the syntax of the SELECT clause is illustrated in the following example:

```
SELECT  [  ALL | DISTINCT  ]
 { single-valued-path-expression  |  aggregation expression  |  OBJECT ( identification-variable )  }
```

The SELECT clause consists of either a single identification variable that is defined in the FROM clause, or a single valued path expression that evaluates to a object reference or CMP value. You can use the DISTINCT keyword to eliminate duplicate references.

For a query that defines a finder method, the query must return an object type consistent with the home that is associated with the finder method. For example, a finder method for a department home can not return employee objects.

Example: SELECT clause

Find all employees that earn more than John:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean e
WHERE  ej.name = 'John'  and e.salary > ej.salary
```

Find all departments that have one or more employees who earn less than 20000:

```
SELECT DISTINCT  e.dept  FROM EmpBean e where e.salary < 20000
```

A select method query can have a path expression that evaluates to an arbitrary value:

```
SELECT  e.dept.name  FROM EmpBean e where e.salary < 2000
```

The previous query returns a collection of name values for those departments having employees earning less than 20000.

A select method query can return an aggregate value:

```
SELECT  avg(e.salary)  FROM EmpBean e
```

Example: Valid dynamic queries

For dynamic queries the syntax is as follows:

```
SELECT { ALL | DISTINCT }  [ selection , ]*  selection
selection  ::= { expression   |  scalar-subselect [[AS] id ] }
```

A scalar-subselect is a subselect that returns a single value.

The following are examples of dynamic queries:

```
SELECT e.name, e.salary+e.bonus as total_pay from EmpBean e
```

```
SELECT  SUM( e.salary+e.bonus) from EmpBean e where e.dept.deptno = ?1
```

ORDER BY clause: The ORDER BY clause specifies an ordering of the objects in the result collection:

```
ORDER BY  [ order_element ,]* order_element
order_element ::= { path-expression | integer } [ ASC | DESC ]
```

The path expression must specify a single valued field that is a primitive type of byte, short, int, long, float, double, char or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp.

ASC specifies ascending order and is the default. DESC specifies descending order.

Integer refers to a selection expression in the SELECT clause.

Example: ORDER BY clause

Return department objects in decreasing deptno order:

```
SELECT  OBJECT(d) FROM  DeptBean d ORDER BY  d.deptno DESC
```

Return employee objects sorted by department number and name:

```
SELECT  OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC,  e.name DESC
```

UNION operation: The UNION clause specifies a combination of the output of two subqueries. The two queries must return the same number of elements and compatible types. For the purposes of UNION, all EJB types in the same inheritance hierarchy are considered compatible. UNION requires that equality be defined for the element types.

```
query_expression := query_term [UNION [ALL] query_term]*

query_term :=  {select_clause_dynamic  from_clause  [where_clause]
 [group_by_clause] [having_clause] } | (query_expression) }
```

You cannot use dependent value objects with UNION.

UNION ALL combines all results together in a single collection.

UNION combines results but eliminates duplicates.

If ORDER BY is used together with UNION, the ORDER BY must refer to selection expression using integer numbers.

Examples: UNION operation

This example returns a collection of all employee objects of type EmpBean and all manager objects of type ManagerBean where ManagerBean is a subtype of EmpBean.

```
select e from EmpBean e union all select m from DeptBean d, in(d.mgr) m
```

This example shows a query that is not valid, because EmpBean and DeptBean are not compatible.

```
select e from EmpBean e union all select d from DeptBean d
```

Subqueries: A subquery can be used in quantified predicates, EXISTS predicate or IN predicate. A subquery should only specify a single element in the SELECT clause. When a path expression appears in a subquery, the identification variable of the path expression must be defined either in the subquery, in one of the containing subqueries, or in the outer query. A scalar subquery is a subquery that returns one value. A scalar subquery can be used in a basic predicate and in the SELECT clause of a dynamic query.

Example: Subqueries

```
SELECT  OBJECT(e) FROM  EmpBean e
WHERE e.salary > ( SELECT  AVG(e1.salary) FROM  EmpBean e1)
```

The above query returns employees who earn more than average salary of all employees.

```
SELECT  OBJECT(e) FROM EmpBean e WHERE  e.salary >
( SELECT AVG(e1.salary) FROM  IN  (e.dept.emps) e1  )
```

The above query returns employees who earn more than average salary of their department.

```
SELECT  OBJECT(e) FROM EmpBean e WHERE e.salary =
( SELECT MAX(e1.salary) FROM IN (e.dept.emps) e1  )
```

The above query returns employees who earn the most in their department.

```
 SELECT OBJECT(e) FROM EmpBean e
WHERE e.salary > ( SELECT AVG(e.salary) FROM EmpBean e1
WHERE YEAR(e1.hireDate) =  YEAR(e.hireDate)  )
```

The above query returns employees who earn more than the average of employees hired in same year.

EJB query language limitations and restrictions: This topic outlines current known limitations and restrictions.

- EJB query language (QL) queries involving enterprise beans with keys made up of relationships to other enterprise beans appear as not valid and cause errors at deployment time. This is a known problem.
- The IBM EJB QL support extends the EJB 2.0 specification in various ways, including relaxing some restrictions, adding support for more DB2 functions, and so on. If portability across various vendor databases or EJB deployment tools is a concern, then care should be taken to write all EJB QL queries strictly according to Chapter 11 in the EJB 2.0 specification.
- Pre-loading across m:n relationships results in the generation of inaccurate structured query language (SQL). This is a known limitation that may be addressed in the future.
- Pre-loading across self referencing relationships causes inaccurate SQL to be generated.
- Avoid relationships between parent and children enterprise beans within the same inheritance hierarchy that are not well-defined.
- EJB Query Language validation for EJB 2.0 JAR files currently runs as a part of the EJB-RDB Mapping validation. If a mapping document (Map.mapxmi file) does not exist in the project, the EJB queries are not validated.

EJB query compatibility issues with SQL: Because an Enterprise JavaBeans query is compiled into SQL, you must be aware of compatibility issues between the Java language and SQL. The two languages differ along the following points that can be critical to correct EJB query formulation:

- The comparison semantics of SQL strings do not exactly match those of the Java language. For example: 'A' (the letter A) and'A ' (the letter A plus a blank space) are considered equal in SQL, but not in the Java language.
- Comparisons and collating order depend on the underlying database. For example, if you are using DB2 with an EBCDIC code page, the collating order is not the same as doing the sort in a Java program. Some databases sort the NULL value low while others sort the NULL value high.
- An arithmetic overflow causes an exception in SQL, but not in the Java language.
- SQL databases have differing minimum and maximum ranges for floating point values, which can differ from floating point value ranges in the Java language. Values near the range limits of Java Double may fail to translate into SQL.
- Java methods do not translate into SQL; therefore standard EJB queries cannot include Java methods.

  Note: Only with the dynamic EJB query service can you use functions that do not translate into SQL. Such functions include Java methods and converters or composers that are used in mapping enterprise beans to relational databases (RDBs). A standard finder or select query that uses any of these functions fails at deployment time with the message ″Cannot push down query″. (You can resolve this problem by changing either the query or the mapping.) The dynamic query run time, however, processes the query by performing the operation involving the function in the application server.

Database restrictions for EJB query: **General database restriction**

All of the enterprise beans involved in a given query must map to the same data source. The EJB query does not support cross-data source join operations.

Specific database restrictions

Different database products place different restrictions on elements that can be included in EJB query statements. Following is a list of those restrictions; check with your database administrator to see if any apply in your environment:

- Certain functions are used in queries that run against DB2 only, because these functions are not supported by other databases. These functions include date and time arithmetic expressions, certain scalar functions (those *not* listed as portable across vendors), and implied scalar functions when used for mapping certain CMP fields. For example, consider mapping an int numeric type to a decimal (5,2) type field. When deployed against a database other than DB2, a finder or select query that contains a CMP field with this particular mapping fails, producing a `Cannot push down query` error message.
- A CMP of type String, when mapped to a character large object (CLOB) in the database, cannot be used in comparison operations because the database does not support CLOB comparisons.
- Databases can impose limits on the length of string values that are used either as literals or input parameters with comparison operators. These limits can hinder query performance. For example: For DB2 on the z/OS platform, the search ″name = ?1″ can fail if the value of ?1 at run time is greater than 255 in length.
- Mapping a numeric CMP type to a column that contains a dissimilar type can cause unexpected results. For example, consider the case of mapping the int numeric type to a column of type decimal (5,2). This scenario does not preserve an exact decimal value (for example, the value 12.25) over the course of transfer from the database to the enterprise bean CMP field, and back again to the database. This mapping causes replacement of the initial value with a whole number (in this case, 12). Consequently, you want to avoid using the CMP field in comparison operations when the CMP field uses a mapping of this nature.
- Some databases do not support a datatype that corresponds to the semantics of java.sql.Time. For example: If a CMP field of type java.sql.Time is mapped to an Oracle DATE column, comparisons on time might not produce the expected result because the year-month-day portion of the column value is truncated in the mapping.

- Some databases treat a zero length string value ('') as a null value; this approach can affect the query results. For the sake of portability, avoid the use of zero length string values.
- Some databases perform division between two integer values using integer arithmetic rules, while others use non-integer rules. This discrepancy might not be desirable in environments that use both kinds of databases. For the sake of portability, avoid the division of integer values in an EJB query.
- Current releases of UDB DB2 for i5/OS only support a TIMESTAMP value of the format 'yyyy-mm-dd-hh.mm.ss.nnnnnn'. This is not compatible with the standard format supported by the java.sql.Timestamp class, which is 'yyyy-mm-dd-hh mm.ss.nnnnnn'. The TIMESTAMP scalar function should be used to convert a string representation of a java.sql.Timestamp object to a value that can be recognized by DB2 UDB for i5/OS.

Rules for data type manipulation in EJB query: **Rules on CMP field type**

You can use a CMP field of any type in a SELECT clause. You must, however, use fields of only the following types in search conditions and in grouping or ordering operations:

- Primitive types: byte, short, int, long, float, double, boolean, char
- Object types: Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.util.Date
- JDBC types: java.sql.Date, java.sql.Time, java.sql.Timestamp
- Binary string: byte[]

Converters and basic types

If ALL of the following conditions occur:
- a CMP field of one of the basic types listed previously is mapped to an SQL column using a converter
- the CMP field appears in the left hand side of a basic predicate
- the right hand side of the predicate is a literal or input parameter

then the toData() method of the converter is used to compute the SQL search value.

For example, given a converter that maps the integer value 10 to the string value ″Ten,″ the following EJB query:

```
e.cmp = 10
```

is translated into the following SQL query:

```
column = 'Ten'
```

If you include a more complicated predicate, such as in the following example:

```
e.cmp * 10 >  e.salary
```

in a finder or select query, you receive the `Cannot push down query` error message. Use the dynamic EJB query service for such multi-function queries; the dynamic query run time processes the predicate in the application server.

Overall, converters preserve equality, collating sequence, and NULL values. If a converter does not meet these requirements, avoid using it for CMP field comparison operations.

User types, converters, and composers

A user type cannot be used in a comparison operation or expression. You can, however, use subfields of the user type in a path expression. For example, consider the CMP addr field with the type `com.exam.Address`, and street, city, and state subfields. The following syntax for a query on this CMP field is not valid:

```
e.addr =  ?1
```

However, a query that designates one of the subfields is valid:

```
e.addr.street = ?1
```

A CMP field can be mapped to an SQL column using Java serialization. Using the CMP field in predicates or expressions for deployment queries usually results in the `Cannot push down query` error message. The dynamic query run time processes the expression by reading and deserializing all instances of the user type in the application server.

However, this expensive process sacrifices performance. You can maintain performance by using a composer in a deployment EJB query. In the previous example, if you want to map the addr field to a binary type, you use a composer to map each subfield to a binary column in the database.

EJB query: Reserved words:

The following words are reserved in WebSphere EJB query:

all, as, distinct, empty, false, from, group, having, in, is, like, select, true, union, where

Avoid using identifiers that start with underscore (for example, _integer) as these are also reserved.

EJB query: BNF syntax:

```
EJB QL ::= [select_clause] from_clause [where_clause] [order_by_clause]

DYNAMIC EJB QL := query_expression  [order_by_clause]

query_expression := query_term [UNION [ALL] query_term]*

query_term :=  {select_clause_dynamic  from_clause  [where_clause]
 [group_by_clause] [having_clause] } | (query_expression) }  [order_by_clause]


from_clause::=FROM identification_variable_declaration
               [, {identification_variable_declaration | collection_member_declaration } ]*

identification_variable_declaration::=collection_member_declaration |
       range_variable_declaration [join]*

join :=  [ { LEFT [OUTER] | INNER  }]  JOIN  {collection_valued_path_expression | single_valued_path_expression}
 [AS] identifier

collection_member_declaration::=
       IN ( collection_valued_path_expression ) [AS] identifier

range_variable_declaration::=abstract_schema_name [AS] identifier

single_valued_path_expression ::=
       {single_valued_navigation | identification_variable}. ( cmp_field |
 method  |  cmp_field.value_object_attribute |  cmp_field.value_object_method )
          | single_valued_navigation

single_valued_navigation::=
       identification_variable.[  single_valued_cmr_field.  ]*
            single_valued_cmr_field

collection_valued_path_expression ::=
       identification_variable.[  single_valued_cmr_field.  ]*
           collection_valued_cmr_field

select_clause::= SELECT { ALL | DISTINCT }  {single_valued_path_expression |
                        identification_variable |  OBJECT ( identification_variable) |
        aggregate_functions }

select_clause_dynamic ::= SELECT { ALL | DISTINCT }  [ selection , ]*  selection
```

```
selection  ::= { expression  |  subselect }  [[AS] id ]

order_by_clause::= ORDER BY [ {single_valued_path_expression | integer}  [ASC|DESC],]*
      {single_valued_path_expression | integer}[ASC|DESC]

where_clause::= WHERE conditional_expression

conditional_expression ::= conditional_term |
                           conditional_expression OR conditional_term
conditional_term ::= conditional_factor |
                     conditional_term AND conditional_factor
conditional_factor ::= [NOT] conditional_primary
conditional_primary::=simple_cond_expression | (conditional_expression)

simple_cond_expression ::= comparison_expression | between_expression |
       like_expression | in_expression | null_comparison_expression |
       empty_collection_comparison_expression | quantified_expression |
       exists_expression | is_of_type_expression  |  collection_member_expression

between_expression ::= expression [NOT] BETWEEN expression AND expression

in_expression ::= single_valued_path_expression [NOT] IN
          { (subselect) |  ( [ atom ,]* atom )  }

atom = { string-literal | numeric-constant | input-parameter }

like_expression ::= expression [NOT] LIKE
                {string_literal | input_parameter}
                [ESCAPE {string_literal | input_parameter}]

null_comparison_expression ::=
       single_valued_path_expression IS [ NOT ] NULL

empty_collection_comparison_expression ::=
       collection_valued_path_expression IS [NOT] EMPTY

collection_member_expression  ::=
       { single_valued_path_expression |  input_paramter }  [ NOT ] MEMBER [ OF ]
         collection_valued_path_expression

quantified_expression ::=
       expression comparison_operator  {SOME | ANY | ALL} (subselect)

exists_expression ::= EXISTS {collection_valued_path_expression |  (subselect)}

subselect ::= SELECT [{ ALL | DISTINCT }]  expression  from_clause [where_clause]
       [group_by_clause] [having_clause]

group_by_clause::= GROUP BY [single_valued_path_expression,]*
                   single_valued_path_expression

having_clause ::= HAVING conditional_expression


is_of_type_expression ::= identifier  IS OF TYPE
            ([[ONLY] abstract_schema_name,]* [ONLY] abstract_schema_name)

comparison_expression ::=  expression   comparison_operator { expression |  ( subquery ) }

comparison_operator ::=    = | > | >= | < | <= | <>

method ::=  method_name( [[expression , ]* expression ] )

expression ::= term |   expression {+|-} term

term ::=  factor |  term {*|/} factor
```

```
factor ::= {+|-} primary

primary ::= single_valued_path_expression | literal |
       ( expression ) |  input_parameter | functions | aggregate_functions

aggregate_functions :=
       AVG([ALL|DISTINCT] expression) |
       COUNT({[ALL|DISTINCT] expression | * | identification_variable }) |
       MAX([ALL|DISTINCT] expression) |
       MIN([ALL|DISTINCT] expression) |
       SUM([ALL|DISTINCT] expression) |

functions ::=
        ABS(expression) |
        BIGINT(expression) |
        CHAR({expression [,{ISO|USA|EUR|JIS}] )  |
        CONCAT (expression , expression ) |
        DATE(expression) |
        DAY({expression ) |
        DAYS( expression) |
        DECIMAL( expression [,integer[,integer]])
        DIGITS( expression) |
        DOUBLE( expression ) |
        FLOAT( expression) |
        HOUR ( expression ) |
        INTEGER( expression ) |
        LCASE ( expression) |
        LENGTH(expression) |
        LOCATE( expression, expression [, expression] ) |
        MICROSECOND( expression ) |
        MINUTE ( expression ) |
        MOD ( expression , expression ) |
        MONTH( expression ) |
        REAL( expression) |
        SECOND( expression ) |
        SMALLINT( expression )  |
        SQRT (  expression) |
        SUBSTRING( expression, expression[, expression]) |
        TIME( expression ) |
        TIMESTAMP( expression ) |
        UCASE ( expression) |
        YEAR( expression )

xrel := XREL identification_variable . { single_valued_cmr_field | collection_valued_cmr_field }
             [ , identification_variable . { single_valued_cmr_field | collection_valued_cmr_field } ]*
```

Comparison of EJB 2.1 specification and WebSphere query language: WebSphere Application Server Version supports the following extensions to the Enterprise JavaBeans Query Language.

Item	
Delimited identifiers	
Dependent Value object attributes used in path expressions	
EJB Inheritance	
EXISTS predicate	
Java methods: EJB bean methods or value object methods	dynamic query only
Multiple element select clauses	dynamic query only
SQL Date/time expressions	
Subqueries, group by, and having clauses	

Using the dynamic query service

Following are common reasons for using the dynamic query service rather than the regular EJB query service (which can be referred to as *deployment query*):

- You need to programmatically define a query at application run time, rather than at deployment.
- You need to return multiple CMP or CMR fields from a query. (Deployment queries allow only a single element to be specified in the SELECT clause.) For more information, see the Example: EJB queries article.
- You want to return a computed expression in the query.
- You want to use value object methods or bean methods in the query statement. For more information, see Path expressions.
- You want to interactively test an EJB query during development, but do not want to repeatedly deploy your application each time you update a finder or select query.

The dynamic query API is a stateless session bean; using it is similar to using any other J2EE EJB application bean. You can consult the API specifications in Reference: Generated API documentation (the section for package com.ibm.websphere.ejbquery).

The dynamic query bean has both a remote and a local interface. If you want to return remote EJB references from the query, or if the query statement contains remote methods, you must use the query remote interface:

remote interface = `com.ibm.websphere.ejbquery.Query`
remote home interface = `com.ibm.websphere.ejbquery.QueryHome`

If you want to return local EJB references from the query, or if the query statement contains local methods, you must use the query local interface:

local interface = `com.ibm.websphere.ejbquery.QueryLocal`
local home interface = `com.ibm.websphere.ejbquery.QueryLocalHome`

Because it uses less application server memory, the local interface ensures better overall EJB performance than the remote.

1. Verify that the query.ear application file is installed on the application server on which your application is to run, if that server is different from the default application server created during installation of the product.

   The `query.ear` file is located in the *app_server_root* directory, where `<WAS_HOME>` is the location of the WebSphere Application Server. The product installation program installs the `query.ear` file on the default application server using a JNDI name of

   ```
   com/ibm/websphere/ejbquery/Query
   ```

   (You or the system administrator can change this name.)
2. Set up authorization for the methods executeQuery(), prepareQuery(), and executePlan() in the remote and local dynamic query interfaces to control access to sensitive data. (This step is necessary only if your application requires security.)

   Because you cannot control which ASN names, CMP fields, or CMR fields can be used in a dynamic EJB query, you or your system administrator must place restrictions on use of the methods. If, for example, a user is permitted to run the executeQuery method, he or she can run any valid dynamic query. In a production environment, you certainly want to restrict access to the remote query interface methods.
3. Write the dynamic query as part of your application client code. You can consult the following examples as query models; they illustrate which import statements to use, and so on:

- Remote interface dynamic query example
- Local interface dynamic query example

4. If the CMP you want to query is on a different module, you should:
   a. do a remote lookup on query.ear
   b. map the query.ear file to the server that the queried CMP bean is installed on.
5. Compile and run your client program with the file **qryclient.jar** in the classpath.

Example: Dynamic query remote interface

When you run a dynamic EJB query using the remote interface, you are calling the executeQuery method on the Query interface. The executeQuery method has a transaction attribute of REQUIRED for this interface; therefore you do not need to explicitly establish a transaction context for the query to run.

Begin with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryHome;
import com.ibm.websphere.ejbquery.Query;
import com.ibm.websphere.ejbquery.QueryIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < 50000";
```

Create a Query object by obtaining a reference from the QueryHome class. (This class defines the executeQuery method.) Note that for the sake of simplicity, the following example uses the dynamic query JNDI name for the Query object:

```
InitialContext ic =  new InitialContext();

Object obj =  ic.lookup("com/ibm/websphere/ejbquery/Query");

QueryHome  qh =
 ( QueryHome) javax.rmi.PortableRemoteObject.narrow( obj, QueryHome.class );
Query qb = qh.create();
```

You then must specify a maximum size for the query result set, which is defined in the QueryIterator object. (See *Class QueryIterator* in Reference: Generated API documentation for more details.) This example sets the maximum size of the result set to 99:

```
QueryIterator it = qb.executeQuery(query, null, null ,0, 99 );
```

The iterator contains a collection of IQueryTuple objects, which are records of the return collection values. (See *Class IQueryTuple* in Reference: Generated API documentation for more details.) Corresponding to the criteria of our example query statement, each tuple in this scenario contains one value of *name* and one value of *object(e)*. To display the contents of this query result, use the following code:

```
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 Emp e = ( Emp) javax.rmi.PortableRemoteObject.narrow( tuple.getObject(2), Emp.class );
 System.out.println( e.getPrimaryKey().toString());
}
```

The output from the program might look something like the following:

```
name Bob
emp 1001
name Dave
emp 298003
...
```

Finally, catch and process any exceptions. An exception might occur because of a syntax error in the query statement or a run-time processing error. The following example catches and processes these exceptions:

```
} catch (QueryException qe) {
    System.out.println("Query Exception "+ qe.getMessage() );
}
```

Handling large result collections for the remote interface query

If you intend your query to return a large collection, you have the option of programming it to return results in multiple smaller, more manageable quantities. Use the skipRow and maxRow parameters on the remote executeQuery method to retrieve the answer in chunks. For example:

```
int skipRow=0;
int maxRow=100;
QueryIterator it = null;
do {
 it = qb.executeQuery(query, null, null ,skipRow, maxRow );
 while (it.hasNext() ) {
 // display result
 skipRow = skipRow + maxRow;
}
} while ( ! it.isComplete() ) ;
```

Example: Dynamic query local interface

When you run a dynamic EJB query using the local interface, you are calling the executeQuery method on the QueryLocal interface. This interface does not initiate a transaction for the method; therefore you must explicitly establish a transaction context for the query to run.

Note: To establish a transaction context, the following example calls the begin() and commit() methods. An alternative to using these methods is simply embedding your query code within an EJB method that runs within a transaction context.

Begin your query code with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryLocalHome;
import com.ibm.websphere.ejbquery.QueryLocal;
import com.ibm.websphere.ejbquery.QueryLocalIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name, object(e) from EmpBean e where e.salary < 50000 ";
```

Create a QueryLocal object by obtaining a reference from the QueryLocalHome class. (This class defines the executeQuery method.) Note that in the following example, `ejb/query` is used as a local EJB reference pointing to the dynamic query JNDI name (`com/ibm/websphere/ejbquery/Query`):

```
InitialContext ic =  new InitialContext();
   QueryLocalHome  qh =  ( LocalQueryHome) ic.lookup( "java:comp/env/ejb/query" );
QueryLocal qb = qh.create();
```

The last portion of code initiates a transaction, calls the executeQuery method, and displays the query results. The QueryLocalIterator class is instantiated because it defines the query result set. (See *Class QueryIterator* in Reference: Generated API documentation for more details.) Keep in mind that the iterator loses validity at the end of the transaction; you must use the iterator in the same transaction scope as the executeQuery call.

```
userTransaction.begin();
QueryLocalIterator it = qb.executeQuery(query, null, null);
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 EmpLocal e = ( EmpLocal ) tuple.getObject(2);
 System.out.println( e.getPrimaryKey().toString());
}
userTransaction.commit();
```

In most situations, the QueryLocalIterator object is *demand-driven*. That is, it causes data to be returned incrementally: for each record retrieval from the database, the next() method must be called on the iterator. (Situations can exist in which the iterator is not demand-driven. For more information, consult the ″Local query interfaces″ subsection of the Dynamic query performance considerations topic.)

Because the full query result set materializes incrementally in the application server memory, you can easily control its size. During a test run, for example, you may decide that return of only a few tuples of the query result is necessary. In that case you should use a call of the close() method on the QueryLocalIterator object to close the query loop. Doing so frees SQL resources that the iterator uses. Otherwise, these resources are not freed until the full result set accumulates in memory, or the transaction ends.

Dynamic query performance considerations

General performance considerations

Use of the following elements in your dynamic query can diminish application performance somewhat:

- Datatype converters and Java methods

  Why: In general, query operations and predicates are translated into SQL so that the database server can perform them. If your query includes datatype converters (for EJB to RDB mapping, for example) or Java methods, however, the associated predicates and operations of your query must be performed in the memory of the application server.
- EJB methods and criteria that call for the return of EJB references

  Why: Queries that incorporate these elements trigger full activation of EJBs in the memory of the application server. (Returning a list of CMP fields from a query does not cause an EJB to be activated.)

When assessing application performance, you should also be aware that dynamic queries share connections with the persistence manager. Consequently, an application that includes a mixture of finder methods, CMR navigation, and dynamic queries relies on a single shared connection between the persistence manager and the dynamic query service to perform these tasks.

Limiting the return collection size

- **Remote interface queries**: The QueryIterator class of the remote interface mandates that all of your query results materialize in application server memory over the course of one method call. The SQL cursor(s) used to run the EJB query are closed upon completion of that call. Because this requirement poses a high risk for creating bottlenecks within the database server, you need to limit the size of any potentially large result collections.

- **Local interface queries**: In most situations, the QueryLocalIterator object behaves as a wrapper around an SQL cursor. It is *demand-driven*; it causes data to be returned incrementally. For each record retrieval from the database, the next() method must be called on the iterator.

  Use of certain operations in local interface queries, however, overrides the demand-driven behavior. In these cases, the query results fully materialize in memory just as do the result collections of remote interface queries. An example of such a case is:

  ```
  select  e.myBusinessMethod( ) from EmpBean e
   where e.salary < 50000 order by 1 desc
  ```

  This query requires performance of an EJB method to produce the final result collection. Consequently, the full dataset from the database must be returned in one collection to application server memory, where the EJB method can be run on the dataset in its entirety. For that reason, local interface query operations that invoke EJB methods are generally not demand-driven. You cannot control the return collection size for such queries.

  Because they *are* demand-driven, all other local interface queries allow you to control the size of return collections. You can use a call of the close() method on the QueryLocalIterator object to close the query loop after the desired number of return values has been fetched from the datastore. Otherwise, the SQL cursor(s) used to run the EJB query are not closed until the full result set accumulates in memory, or the transaction ends.

Access intent implications for dynamic query

WebSphere Application Server gives you the option to set access intent policies for your entity enterprise beans as a way of managing their transfer of data with the underlying datastore. An access intent policy controls the isolation level used on the data source connection, as well as the database locks used during data retrieval. By manipulating these elements, you can maximize the efficiency of your application's data flow. To learn more, begin with the topics "Access intent policies" on page 204 and "Concurrency control" on page 204.

When formulating dynamic queries, keep in mind the following considerations concerning their interaction with access intent policies:

- A dynamic query uses the first ASN name in the FROM clause to determine access intent.
- The collection increment attribute of an access intent policy is not used in processing a dynamic query.
- When performed on entity beans that have a pessimistic-Update access intent policy, your dynamic queries must return updateable collections. Therefore you need to formulate your query statements to return only collections of entity beans, *not* collections of CMP fields. For example, the statement `select object(c) from Customer` is valid for a dynamic query performed under the constraint of a pessimistic-Update policy. The statement `select c.name from Customer c`, however, is not a valid dynamic query under this constraint.
- Using pessimistic-Update policy places restrictions on the types of query expressions. The restrictions depend on the back end database type and release. Refer to the topic Access intent -- isolation levels and update locks for details.

Dynamic query API: prepareQuery() and executePlan() methods

Use these methods to more efficiently allocate the overhead associated with dynamic query. They are equivalent in function to the prepareStatement() and executeQuery() methods of the JDBC API.

To perform a dynamic EJB query, the application server must parse the query string into SQL at run time. You can, of course, eliminate run-time overhead by choosing to perform a standard EJB query instead of a dynamic query. Sometimes referred to as *deployment queries*, standard queries are parsed and built at deployment, then performed by a finder or select method.

Another option is to write code that redistributes dynamic query overhead for better application performance. Begin by calling the prepareQuery() method in place of the executeQuery() method. The prepareQuery() method parses and translates your query, and returns a string called a *query plan*. The plan contains the SQL statement produced by parsing and translation, as well as other information needed

by the dynamic query API. Save this string in your application and call the executePlan() method with the string to run your query. (You also might want to use the prepareQuery() method simply to see the SQL translation product; just call the method and display the return value.)

Pass the parameters of your query as an array of type Object on the prepareQuery() and the executePlan() method calls. Ensure that you pass appropriate data types, because the application server validates your query according to parameter type (rather than actual values) when it processes the prepareQuery() method call.

Example code

Note: In the example code that follows, the first executePlan() method call substitutes `parms[0]` for `?1`. Hence the first query performed is functionally equivalent to the following query statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000
```

The second call runs a query that is functionally equivalent to this statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 60000
```

The example:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < ?1";
QueryIterator it = null;
Integer[] parms = new Integer[1];
parms[0] = new Integer(0);
```

In the call to `prepareQuery()`, pass any Integer value. Doing so defines `?1` as an Integer type, as in the following:

```
String queryPlan= qb.prepareQuery(query, parms, null );

   parms[0] = new Integer(50000);
```

Next you run the query with a real value of `Integer(50000)` for `?1`:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000it =
 qb.executePlan( queryPlan, parms, 0, 99);

parms[0] = new Integer(60000);
```

Run the query again with a different value of `Integer(60000)` for `?1`:

```
it = qb.executePlan( queryPlan, parms, 0, 99);
```

Comparison of the dynamic and deployment EJB query services

You can use the dynamic query service to build and execute queries against entity beans constructed dynamically at run time, rather than defining them at deployment time. By using dynamic query you gain the flexibility of queries defined at run time and utilize the power of EJB-Query Language (QL). Apart from supporting all of the capabilities of an EJB-QL query, dynamic query adds functionality not available to standard static query. Two examples are the ability to select multiple data fields directly from the bean itself (static queries currently only allow one) and executing business methods directly in the query.

You can effectively create more efficient and less resource intensive applications with dynamic query. For example, two data fields are required from the results of a query. Because a standard EJB-QL query can only select one data field, it is necessary to select the entire EJB object and extract the needed data from the returned results through data access methods, possibly traversing Container Managed Relationships (CMR) boundaries in the process. However, when using dynamic query, you can get both pieces of data directly from the query without additional CMR traversal or accessor methods. This principle is the key to evaluating whether or not dynamic query can be used for performance gain. You should review the amount of data that must be retrieved, in addition to the amount of business logic needed to retrieve it, for example, CMR traversal or accessor methods.

Using parameters in the query rather than literal values is another performance consideration. Under most circumstances, it is better to define conditional values as parameters in the query and then pass those parameters through the appropriate mechanisms. By using this method, you have a greater chance of matching a cached query plan, and you eliminate the need to parse and build the plan from scratch. For example, ″SELECT Object(o) FROM schemaname AS o WHERE o.fieldname LIKE foo″, is more appropriately expressed as ″SELECT Object(o) FROM schemaname AS o WHERE o.fieldname LIKE ?1″ with the value *foo* passed as a parameter to the executeQuery method. The result is that any subsequent execution of a dynamic query structure that is the same, except for different string literal conditions, is registered as a plan cache hit (which delivers better ″observed″ performance).

When used as a direct replacement for an equivalent static query, dynamic query is approximately 25% slower than the static variation. This slowdown is due to the need for parsing and building a plan for the query, in addition to executing it. In the static variation, these costs are paid at deploy time. Despite this, the added functionality gained through the use of dynamic query, specifically the ability to select multiple data fields in a single query even across CMRs, creates opportunities to utilize dynamic query for the sake of performance improvement.

Using the dynamic query service

Following are common reasons for using the dynamic query service rather than the regular EJB query service (which can be referred to as *deployment query*):

- You need to programmatically define a query at application run time, rather than at deployment.
- You need to return multiple CMP or CMR fields from a query. (Deployment queries allow only a single element to be specified in the SELECT clause.) For more information, see the Example: EJB queries article.
- You want to return a computed expression in the query.
- You want to use value object methods or bean methods in the query statement. For more information, see Path expressions.
- You want to interactively test an EJB query during development, but do not want to repeatedly deploy your application each time you update a finder or select query.

The dynamic query API is a stateless session bean; using it is similar to using any other J2EE EJB application bean. You can consult the API specifications in Reference: Generated API documentation (the section for package com.ibm.websphere.ejbquery).

The dynamic query bean has both a remote and a local interface. If you want to return remote EJB references from the query, or if the query statement contains remote methods, you must use the query remote interface:

```
remote interface = com.ibm.websphere.ejbquery.Query
remote home interface = com.ibm.websphere.ejbquery.QueryHome
```

If you want to return local EJB references from the query, or if the query statement contains local methods, you must use the query local interface:

```
local interface = com.ibm.websphere.ejbquery.QueryLocal
local home interface = com.ibm.websphere.ejbquery.QueryLocalHome
```

Because it uses less application server memory, the local interface ensures better overall EJB performance than the remote.

1. Verify that the query.ear application file is installed on the application server on which your application is to run, if that server is different from the default application server created during installation of the product.

The `query.ear` file is located in the *app_server_root* directory, where `<WAS_HOME>` is the location of the WebSphere Application Server. The product installation program installs the `query.ear` file on the default application server using a JNDI name of

```
com/ibm/websphere/ejbquery/Query
```

(You or the system administrator can change this name.)

2. Set up authorization for the methods executeQuery(), prepareQuery(), and executePlan() in the remote and local dynamic query interfaces to control access to sensitive data. (This step is necessary only if your application requires security.)

   Because you cannot control which ASN names, CMP fields, or CMR fields can be used in a dynamic EJB query, you or your system administrator must place restrictions on use of the methods. If, for example, a user is permitted to run the executeQuery method, he or she can run any valid dynamic query. In a production environment, you certainly want to restrict access to the remote query interface methods.
3. Write the dynamic query as part of your application client code. You can consult the following examples as query models; they illustrate which import statements to use, and so on:
   - Remote interface dynamic query example
   - Local interface dynamic query example
4. If the CMP you want to query is on a different module, you should:
   a. do a remote lookup on query.ear
   b. map the query.ear file to the server that the queried CMP bean is installed on.
5. Compile and run your client program with the file **qryclient.jar** in the classpath.

Example: Dynamic query remote interface

When you run a dynamic EJB query using the remote interface, you are calling the executeQuery method on the Query interface. The executeQuery method has a transaction attribute of REQUIRED for this interface; therefore you do not need to explicitly establish a transaction context for the query to run.

Begin with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryHome;
import com.ibm.websphere.ejbquery.Query;
import com.ibm.websphere.ejbquery.QueryIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < 50000";
```

Create a Query object by obtaining a reference from the QueryHome class. (This class defines the executeQuery method.) Note that for the sake of simplicity, the following example uses the dynamic query JNDI name for the Query object:

```
InitialContext ic =  new InitialContext();

Object obj =  ic.lookup("com/ibm/websphere/ejbquery/Query");

QueryHome  qh =
 ( QueryHome) javax.rmi.PortableRemoteObject.narrow( obj, QueryHome.class );
Query qb = qh.create();
```

You then must specify a maximum size for the query result set, which is defined in the QueryIterator object. (See *Class QueryIterator* in Reference: Generated API documentation for more details.) This example sets the maximum size of the result set to 99:

```
QueryIterator it = qb.executeQuery(query, null, null ,0, 99 );
```

The iterator contains a collection of IQueryTuple objects, which are records of the return collection values. (See *Class IQueryTuple* in Reference: Generated API documentation for more details.) Corresponding to the criteria of our example query statement, each tuple in this scenario contains one value of *name* and one value of *object(e)*. To display the contents of this query result, use the following code:

```
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 Emp e = ( Emp) javax.rmi.PortableRemoteObject.narrow( tuple.getObject(2), Emp.class );
 System.out.println( e.getPrimaryKey().toString());
}
```

The output from the program might look something like the following:

```
name Bob
emp 1001
name Dave
emp 298003
...
```

Finally, catch and process any exceptions. An exception might occur because of a syntax error in the query statement or a run-time processing error. The following example catches and processes these exceptions:

```
} catch (QueryException qe) {
    System.out.println("Query Exception "+ qe.getMessage() );
}
```

Handling large result collections for the remote interface query

If you intend your query to return a large collection, you have the option of programming it to return results in multiple smaller, more manageable quantities. Use the skipRow and maxRow parameters on the remote executeQuery method to retrieve the answer in chunks. For example:

```
int skipRow=0;
int maxRow=100;
QueryIterator it = null;
do {
 it = qb.executeQuery(query, null, null ,skipRow, maxRow );
 while (it.hasNext() ) {
 // display result
 skipRow = skipRow + maxRow;
}
} while ( ! it.isComplete() ) ;
```

Example: Dynamic query local interface

When you run a dynamic EJB query using the local interface, you are calling the executeQuery method on the QueryLocal interface. This interface does not initiate a transaction for the method; therefore you must explicitly establish a transaction context for the query to run.

Note: To establish a transaction context, the following example calls the begin() and commit() methods. An alternative to using these methods is simply embedding your query code within an EJB method that runs within a transaction context.

Begin your query code with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryLocalHome;
import com.ibm.websphere.ejbquery.QueryLocal;
import com.ibm.websphere.ejbquery.QueryLocalIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name, object(e) from EmpBean e where e.salary < 50000 ";
```

Create a QueryLocal object by obtaining a reference from the QueryLocalHome class. (This class defines the executeQuery method.) Note that in the following example, `ejb/query` is used as a local EJB reference pointing to the dynamic query JNDI name (`com/ibm/websphere/ejbquery/Query`):

```
InitialContext ic =  new InitialContext();
   QueryLocalHome  qh =  ( LocalQueryHome) ic.lookup( "java:comp/env/ejb/query" );
QueryLocal qb = qh.create();
```

The last portion of code initiates a transaction, calls the executeQuery method, and displays the query results. The QueryLocalIterator class is instantiated because it defines the query result set. (See *Class QueryIterator* in Reference: Generated API documentation for more details.) Keep in mind that the iterator loses validity at the end of the transaction; you must use the iterator in the same transaction scope as the executeQuery call.

```
userTransaction.begin();
QueryLocalIterator it = qb.executeQuery(query, null, null);
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 EmpLocal e = ( EmpLocal ) tuple.getObject(2);
 System.out.println( e.getPrimaryKey().toString());
}
userTransaction.commit();
```

In most situations, the QueryLocalIterator object is *demand-driven*. That is, it causes data to be returned incrementally: for each record retrieval from the database, the next() method must be called on the iterator. (Situations can exist in which the iterator is not demand-driven. For more information, consult the ″Local query interfaces″ subsection of the Dynamic query performance considerations topic.)

Because the full query result set materializes incrementally in the application server memory, you can easily control its size. During a test run, for example, you may decide that return of only a few tuples of the query result is necessary. In that case you should use a call of the close() method on the QueryLocalIterator object to close the query loop. Doing so frees SQL resources that the iterator uses. Otherwise, these resources are not freed until the full result set accumulates in memory, or the transaction ends.

Dynamic query performance considerations

General performance considerations

Use of the following elements in your dynamic query can diminish application performance somewhat:

- Datatype converters and Java methods

  Why: In general, query operations and predicates are translated into SQL so that the database server can perform them. If your query includes datatype converters (for EJB to RDB mapping, for example) or Java methods, however, the associated predicates and operations of your query must be performed in the memory of the application server.
- EJB methods and criteria that call for the return of EJB references

Why: Queries that incorporate these elements trigger full activation of EJBs in the memory of the application server. (Returning a list of CMP fields from a query does not cause an EJB to be activated.)

When assessing application performance, you should also be aware that dynamic queries share connections with the persistence manager. Consequently, an application that includes a mixture of finder methods, CMR navigation, and dynamic queries relies on a single shared connection between the persistence manager and the dynamic query service to perform these tasks.

Limiting the return collection size

- **Remote interface queries**: The QueryIterator class of the remote interface mandates that all of your query results materialize in application server memory over the course of one method call. The SQL cursor(s) used to run the EJB query are closed upon completion of that call. Because this requirement poses a high risk for creating bottlenecks within the database server, you need to limit the size of any potentially large result collections.
- **Local interface queries**: In most situations, the QueryLocalIterator object behaves as a wrapper around an SQL cursor. It is *demand-driven*; it causes data to be returned incrementally. For each record retrieval from the database, the next() method must be called on the iterator.

  Use of certain operations in local interface queries, however, overrides the demand-driven behavior. In these cases, the query results fully materialize in memory just as do the result collections of remote interface queries. An example of such a case is:

  ```
  select  e.myBusinessMethod( ) from EmpBean e
   where e.salary < 50000 order by 1 desc
  ```

  This query requires performance of an EJB method to produce the final result collection. Consequently, the full dataset from the database must be returned in one collection to application server memory, where the EJB method can be run on the dataset in its entirety. For that reason, local interface query operations that invoke EJB methods are generally not demand-driven. You cannot control the return collection size for such queries.

  Because they *are* demand-driven, all other local interface queries allow you to control the size of return collections. You can use a call of the close() method on the QueryLocalIterator object to close the query loop after the desired number of return values has been fetched from the datastore. Otherwise, the SQL cursor(s) used to run the EJB query are not closed until the full result set accumulates in memory, or the transaction ends.

Access intent implications for dynamic query

WebSphere Application Server gives you the option to set access intent policies for your entity enterprise beans as a way of managing their transfer of data with the underlying datastore. An access intent policy controls the isolation level used on the data source connection, as well as the database locks used during data retrieval. By manipulating these elements, you can maximize the efficiency of your application's data flow. To learn more, begin with the topics "Access intent policies" on page 204 and "Concurrency control" on page 204.

When formulating dynamic queries, keep in mind the following considerations concerning their interaction with access intent policies:

- A dynamic query uses the first ASN name in the FROM clause to determine access intent.
- The collection increment attribute of an access intent policy is not used in processing a dynamic query.
- When performed on entity beans that have a pessimistic-Update access intent policy, your dynamic queries must return updateable collections. Therefore you need to formulate your query statements to return only collections of entity beans, *not* collections of CMP fields. For example, the statement `select object(c) from Customer` is valid for a dynamic query performed under the constraint of a pessimistic-Update policy. The statement `select c.name from Customer c`, however, is not a valid dynamic query under this constraint.
- Using pessimistic-Update policy places restrictions on the types of query expressions. The restrictions depend on the back end database type and release. Refer to the topic Access intent -- isolation levels and update locks for details.

Dynamic query API: prepareQuery() and executePlan() methods

Use these methods to more efficiently allocate the overhead associated with dynamic query. They are equivalent in function to the prepareStatement() and executeQuery() methods of the JDBC API.

To perform a dynamic EJB query, the application server must parse the query string into SQL at run time. You can, of course, eliminate run-time overhead by choosing to perform a standard EJB query instead of a dynamic query. Sometimes referred to as *deployment queries*, standard queries are parsed and built at deployment, then performed by a finder or select method.

Another option is to write code that redistributes dynamic query overhead for better application performance. Begin by calling the prepareQuery() method in place of the executeQuery() method. The prepareQuery() method parses and translates your query, and returns a string called a *query plan*. The plan contains the SQL statement produced by parsing and translation, as well as other information needed by the dynamic query API. Save this string in your application and call the executePlan() method with the string to run your query. (You also might want to use the prepareQuery() method simply to see the SQL translation product; just call the method and display the return value.)

Pass the parameters of your query as an array of type Object on the prepareQuery() and the executePlan() method calls. Ensure that you pass appropriate data types, because the application server validates your query according to parameter type (rather than actual values) when it processes the prepareQuery() method call.

Example code

Note: In the example code that follows, the first executePlan() method call substitutes `parms[0]` for `?1`. Hence the first query performed is functionally equivalent to the following query statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000
```

The second call runs a query that is functionally equivalent to this statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 60000
```

The example:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < ?1";
QueryIterator it = null;
Integer[] parms = new Integer[1];
parms[0] = new Integer(0);
```

In the call to `prepareQuery()`, pass any Integer value. Doing so defines `?1` as an Integer type, as in the following:

```
String queryPlan= qb.prepareQuery(query, parms, null );

   parms[0] = new Integer(50000);
```

Next you run the query with a real value of `Integer(50000)` for `?1`:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000it =
 qb.executePlan( queryPlan, parms, 0, 99);

parms[0] = new Integer(60000);
```

Run the query again with a different value of `Integer(60000)` for `?1`:

```
it = qb.executePlan( queryPlan, parms, 0, 99);
```

Comparison of the dynamic and deployment EJB query services

You can use the dynamic query service to build and execute queries against entity beans constructed dynamically at run time, rather than defining them at deployment time. By using dynamic query you gain

the flexibility of queries defined at run time and utilize the power of EJB-Query Language (QL). Apart from supporting all of the capabilities of an EJB-QL query, dynamic query adds functionality not available to standard static query. Two examples are the ability to select multiple data fields directly from the bean itself (static queries currently only allow one) and executing business methods directly in the query.

You can effectively create more efficient and less resource intensive applications with dynamic query. For example, two data fields are required from the results of a query. Because a standard EJB-QL query can only select one data field, it is necessary to select the entire EJB object and extract the needed data from the returned results through data access methods, possibly traversing Container Managed Relationships (CMR) boundaries in the process. However, when using dynamic query, you can get both pieces of data directly from the query without additional CMR traversal or accessor methods. This principle is the key to evaluating whether or not dynamic query can be used for performance gain. You should review the amount of data that must be retrieved, in addition to the amount of business logic needed to retrieve it, for example, CMR traversal or accessor methods.

Using parameters in the query rather than literal values is another performance consideration. Under most circumstances, it is better to define conditional values as parameters in the query and then pass those parameters through the appropriate mechanisms. By using this method, you have a greater chance of matching a cached query plan, and you eliminate the need to parse and build the plan from scratch. For example, ″SELECT Object(o) FROM schemaname AS o WHERE o.fieldname LIKE foo″, is more appropriately expressed as ″SELECT Object(o) FROM schemaname AS o WHERE o.fieldname LIKE ?1″ with the value *foo* passed as a parameter to the executeQuery method. The result is that any subsequent execution of a dynamic query structure that is the same, except for different string literal conditions, is registered as a plan cache hit (which delivers better ″observed″ performance).

When used as a direct replacement for an equivalent static query, dynamic query is approximately 25% slower than the static variation. This slowdown is due to the need for parsing and building a plan for the query, in addition to executing it. In the static variation, these costs are paid at deploy time. Despite this, the added functionality gained through the use of dynamic query, specifically the ability to select multiple data fields in a single query even across CMRs, creates opportunities to utilize dynamic query for the sake of performance improvement.

Internationalization

Task overview: Globalizing applications

An application that can present information to users according to regional cultural conventions is said to be *globalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In a globalized application, a user in one region sees error messages, output, and interface elements in the requested language. Date and time formats, as well as currencies, are presented appropriately for users in the specified region. A user in another region sees output in the conventional language or format for that region. Globalization consists of two phases: *internationalization* (enabling an application component to use regional conventions) and *localization* (implementing a specific regional convention). This product supports globalization through the use of its localizable-text API and internationalization service.

- Make sure the server runtime environment is properly configured.

  For more information about supported locales and character encodings, see “Working with locales and character encodings” on page 2031.
- Implement message catalogs in your application by using the localizable-text API.

  This product supports the maintenance and deployment of centralized message catalogs for the output of properly formatted, language-specific (*localized*) interface strings.

  For more information about the localizable-text API, see “Task overview: Internationalizing interface strings (localizable-text API)” on page 2029.
- Implement more extensive locale support by using the internationalization service.

With the internationalization service, you can manage the distribution of the internationalization information, or *internationalization context*, that is necessary to perform localizations within Java 2 Platform, Enterprise Edition (J2EE) application components. Supported application components also include Web service client environments and Web service-enabled enterprise beans.

For more information about the internationalization service, see “Task overview: Internationalizing application components (internationalization service)” on page 2029.

Globalization

An application that can present information to users according to regional cultural conventions is said to be *globalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In a globalized application, a user in one region sees error messages, output, and interface elements in the requested language. Date and time formats, as well as currencies, are presented appropriately for users in the specified region. A user in another region sees output in the conventional language or format for that region. Globalization consists of two phases: *internationalization* (enabling an application component to use regional conventions) and *localization* (implementing a specific regional convention).

Historically, the creation of globalized applications has been restricted to large corporations writing complex systems. However, given the rise in distributed computing and in the use of the World Wide Web, application developers are pressured to globalize a much wider variety of applications. This trend requires making globalization techniques much more accessible to application developers.

Internationalization of an application is driven by two variables, the time zone and the locale. The *time zone* indicates how to compute the local time as an offset from a standard time like Greenwich Mean Time. The *locale* is a collection of information about language, currency, and the conventions for presenting information like dates. A time zone can cover many locales, and a single locale can span time zones. With both time zone and locale, the date, time, currency, and language for users in a specific region can be determined.

A first step: Localization of interface strings

In an application that is not globalized, the user interface is unalterably written into the application code. Internationalizing a user interface adds a layer of abstraction into the design of an application. The additional layer of abstraction enables you to localize the application for each locale that must be supported by the application.

In a localized application, the locale determines the message catalog from which the application retrieves message strings. Instead of printing an error message, the application represents the error message with some language-neutral information; in the simplest case, each error condition corresponds to a key. To print a usable error message, the application looks up the key in a *message catalog*. Each message catalog is a list of keys with associated strings. Different message catalogs provide strings for the different languages that are supported. The application looks up the key in the appropriate catalog, retrieves the corresponding error message in the requested language, and prints the string for the user.

Localization of text can be used for far more than translating error messages. For example, by using keys to represent each element in a graphical user interface (GUI) and by providing the appropriate message catalogs, the GUI (buttons, menus, and so on) can support multiple languages. Extending support to additional languages requires that you provide message catalogs for those languages; in many cases, the application needs no further modification.

The localizable-text package is a set of Java classes and interfaces that can be used to localize the strings in distributed applications easily. Language-specific string catalogs can be stored centrally so that they can be maintained efficiently.

Globalization challenges in distributed applications

With the advent of Internet-based business computational models, applications increasingly consist of clients and servers that operate in different geographical regions. These differences introduce the following challenges to the task of designing a solid client-server infrastructure:

Clients and servers can run on computers that have different endian architectures or code sets

Clients and servers can reside in computers that have different endian architectures: A client can reside in a little-endian CPU, while the server code runs in a big-endian one. A client might want to call a business method on a server running in a code set different from that of the client.

A client-server infrastructure must define precise endian and code-set tracking and conversion rules. The Java platform has nearly eliminated these problems in a unique way by relying on its Java virtual machine (JVM), which encodes all of the string data in UCS-2 format and externalizes everything in big-endian format. The JVM uses a set of platform-specific programs for interfacing with the native platform. These programs perform any necessary code set conversions between UCS-2 and the native code set of a platform.

Clients and servers can run on computers with different locale settings

Client and server processes can use different locale settings. For example, a Spanish client might call a business method upon an object that resides on an American English server. Some business methods are locale-sensitive in nature; for example, given a business method that returns a sorted list of strings, the Spanish client expects that list to be sorted according to the Spanish collating sequence, not in the English collating sequence of the server. Because data retrieval and sorting procedures run on the server, the locale of the client must be available to perform a legitimate sort.

A similar consideration applies in instances where the server has to return strings containing date, time, currency, exception messages, and so on, that are formatted according to the cultural expectations of the client.

Clients and servers can reside in different time zones

Client and server processes can run in different time zones. To date, all internationalization literature and resources concentrate mainly on code set and locale-related issues. They have generally ignored the time zone issue, even though business methods can be sensitive to time zone as well as to locale.

For example, suppose that a vendor makes the claim that orders received before 2:00 PM are processed by 5:00 PM the same day. The times given, of course, are in the time zone of the server that is processing the order. It is important to know the time zone of the client to give customers in other time zones the correct times for same-day processing.

Other time zone-sensitive operations include time stamping messages logged to a server, and accessing file or database resources. The concept of Daylight Savings Time further complicates the time zone issue.

Java 2 Platform, Enterprise Edition (J2EE) provides support for application components that run on computers with differing endian architecture and code sets. It does not provide dedicated support for application components that run on computers with different locales or time zones.

The conventional method for solving locale and time zone mismatches across remote application components is to pass one or more extra parameters on all business methods needed to convey the client-side locale or time zone to the server. Although simple, this technique has the following limitations when used in Enterprise JavaBeans (EJB) applications:

- It is intrusive because it requires that one or more parameters be added to all bean methods in the call chain to locale-sensitive or time zone-sensitive methods.
- It is inherently error-prone.
- It is impracticable within applications that do not support modification, such as legacy applications.

The internationalization service addresses the challenges posed by locale and time zone mismatch without incurring the limitations of conventional techniques. The service systematically manages the distribution of internationalization contexts across the various components of EJB applications, including client applications, enterprise beans, and servlets. For more information, see “Task overview: Internationalizing application components (internationalization service)” on page 2029.

Language versions offered by this product

This product is offered in several languages, as enabled by the operating platform on which the product is installed.

The following language versions are available:

- Brazilian Portuguese
- Chinese (Simplified)
- Chinese (Traditional)
- English
- French
- German
- Italian
- Japanese
- Korean
- Spanish

Globalization: Resources for learning

Use links in this topic to find relevant supplemental information about globalization. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks™ that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- “Programming instructions and examples”
- “Programming specifications”

Programming instructions and examples

- Java internationalization tutorial

  An online tutorial that explains how to use the Java 2 SDK Internationalization API.
- Globalize your On Demand Business

  IBM’s portal site for delivering globalized applications.

Programming specifications

- Java 2 SDK, Standard Edition Documentation: Internationalization

  The Java 1.4.2 internationalization documentation from Sun Microsystems, including a list of supported locales and encodings. For other versions of the Java platform, click the ″Internationalization Home Page″ link on that page.
- Java Specification Request 150, Internationalization Service for J2EE

  The specification of the J2EE internationalization service that is currently being developed through the Java Community Process.
- W3C, Internationalization Core Working Group

The W3C's Internationalization Core Working Group responsible for investigating the internationalization of Web services, in particular, the dependence of Web services on language, culture, region, and locale-related contexts.

- Making the WWW truly World Wide

  The W3C effort to make World Wide Web technology work with the many writing systems, languages, and cultural conventions of the global community:

Task overview: Internationalizing interface strings (localizable-text API)

This topic summarizes the steps involved in implementing message catalogs through the localizable-text API.

This product supports the maintenance and deployment of centralized message catalogs for the output of properly formatted, language-specific (*localized*) interface strings.

1. Identify localizable text in your application.
2. Create the message catalogs that are necessary for the locales to be supported by your application.
3. In your application code, compose the language-specific strings for output.
4. Using an assembly tool, assemble your application code as one or more application components.
5. Prepare the localizable-text package for deployment with your localized application. In this step, you create a deployment Java archive (JAR) file.
6. Assemble the application modules and the deployment JAR file into a Java 2 Platform, Enterprise Edition (J2EE) application.
7. Deploy and manage the application.

Your application is deployed with localized text.

Task overview: Internationalizing application components (internationalization service)

This topic summarizes the steps involved in using the internationalization service.

With the internationalization service, you can manage the distribution of the internationalization information, or *internationalization context*, that is necessary to perform localizations within Java 2 Platform, Enterprise Edition (J2EE) application components. Supported application components also include Web service client environments and Web service-enabled enterprise beans.

1. Use the internationalization context API within application components to obtain or manage internationalization context.

   Servlet and enterprise bean business methods can use internationalization context to perform locale- and time zone-sensitive localizations. Enterprise JavaBeans (EJB) client applications, and server components that are configured to manage internationalization context must use the internationalization context API to set the context elements scoped to their invocations.

   You use the internationalization context API within Web service-enabled J2EE client programs and stateless session beans in the same manner that you would use conventional J2EE components, with one exception. Internationalization context propagated over Web service requests contains a time zone ID, whereas conventional Remote Method Invocation/ Internet Inter-ORB Protocol (RMI/IIOP) requests propagate complete time zone information, including the raw offset, Daylight Savings Time information, and so on.
2. Assemble internationalized applications.

   The internationalization type specifies the internationalization policy that applies to a servlet or an enterprise bean and, in particular, indicates whether the application component or its hosting J2EE container manages internationalization context. Container internationalization attributes can be specified for container-managed servlet and enterprise bean business methods. These attributes tailor a policy by indicating which context the container scopes to an invocation. Configuring

internationalization policies declaratively prescribes, by means of the application deployment descriptor, the distribution and management of context throughout an application.

As you edit the deployment descriptor for assembly, you can also set the internationalization type and configure any container internationalization attributes for the servlets and enterprise beans in your application.

You configure internationalization type and container internationalization attributes for Web service-enabled stateless session beans in the same manner as you do for conventional beans.

3. Manage the internationalization service.

   Use the administrative console to enable the service on all application servers.

   By default, the service is enabled within J2EE client environments but is disabled on application servers. You must enable the service on all application servers hosting your servlets and enterprise beans to use internationalization context.

4. Troubleshoot the internationalization service as needed.

   Use the administrative console to enable the trace service to log internationalization service messages when debugging your applications.

   The trace strings for the internationalization service follow; use both:

   ```
   com.ibm.ws.i18n.context.*=all=enabled:com.ibm.websphere.i18n.context.*=all=enabled
   ```

Internationalization service

In a distributed client-server environment, application processes can run on different machines, configured for different locales, corresponding to different cultural conventions; they can also be located across geographical boundaries. The internationalization service can help manage your application in a globally distributed environment.

For an understanding of how differences in locale impact application development, read “Globalization” on page 2026.

Java 2 Platform, Enterprise Edition (J2EE) provides support for application components that run on computers with differing endian architecture and code sets. It does not provide dedicated support for application components that run on computers with different locales or time zones.

The internationalization service addresses the challenges posed by locale and time zone mismatch without incurring the limitations of conventional techniques. The service systematically manages the distribution of internationalization contexts across the various components of EJB applications, including client applications, enterprise beans, and servlets.

The service works by associating an internationalization context with every service request within an application. When a client-side component calls a business method, the internationalization service interposes by obtaining the internationalization context associated with the current client-side process and by attaching that context to the outgoing request. On the server side, the internationalization service again interposes by detaching the context from the incoming request and associating it with the server-side process on which the business method will run, effectively scoping the context to the business method. For HTTP requests, the caller context is constructed from the HTTP attributes and default values. The service propagates internationalization context on subsequent business method invocations in the same manner, which distributes the context of the originating request over the entire chain of business method invocations.

This basic operation of scoping and propagation is defined precisely by *internationalization context management policies*. Internationalization policies specify whether an application component or its hosting J2EE container are to manage internationalization context. For container-managed components, the policy indicates which internationalization context the container scopes to invocations on that component. Server components configured to manage internationalization context, as well as EJB clients, must use the internationalization context API to manage the internationalization context elements scoped to their invocations.

Every application component has a default policy, which can be overridden and tailored for servlets and enterprise beans at assembly time.

At run time, application components can use the internationalization context API to get any element of the internationalization contexts scoped to an invocation. To programmatically access context elements, application components first resolve an internationalization context API reference, then call the appropriate API method to access the various context elements, such as the caller locale or the invocation time zone. These elements can be used in calls to Java 2 SDK internationalization API methods; for example, to perform localizations such as formatting messages, configuring dates, or comparing strings.

Working with locales and character encodings

Internationalization support for this product relies on that provided by the Java 2 Platform, Standard Edition (J2SE). Support varies by platform.

- Verify that the operating system on which the application server is installed supports the locales and encodings that you plan to use.

  Java internationalization support might use underlying services of the operating system. For example, if user IDs for your server are expected to contain non-English characters, make sure that the operating system is configured to process those characters.
- Plan for encoding changes as necessary.

  Consider differences in encoding support among operating system subcomponents. Although this product and the Java platform are based on Unicode encoding, it is not always possible to run applications in a purely Unicode environment.
- Set the console.encoding property as necessary.

If your application produces an UnsupportedEncodingException exception, check your operating system documentation to determine if the target operating system supports the required encoding and adjust the runtime environment as needed.

Windows

For example, on the Windows® platform, the command prompt runs in a Windows code page. Not all Windows code pages are supported by the Java platform, so it is possible to get a Java exception when running a command-line program, such as wsadmin, in an unsupported code page. To avoid exceptions, use the chcp command to explicitly set the code page to an encoding that is supported by the Java platform.

- Before command-line calls, change the code page.

  For example, Arabic code page 720 is not supported by the Java platform, but the Arabic code page for Windows (Cp1256) systems is. Type `chcp 1256`
- When starting a localized application from a command prompt, set the console.encoding property.

  For Arabic, pass the following parameter: `-Dconsole.encoding=Cp1256`

Administering the internationalization service

To use internationalization context in an Enterprise JavaBeans (EJB) application, the internationalization service must be enabled in the runtime environments for all server-side components (servlets and enterprise beans, including session beans enabled for Web service usage) as well as all client-side components (EJB client applications and Web service clients).

If you do not require the internationalization service, do not enable it. Leaving the service disabled prevents any possible performance degradation incurred by the implicit distribution of internationalization resources.

The internationalization service cannot be enabled for HTTP clients, because support for internationalization in that case is provided by the browser, not by the application server.

- Enable or disable the internationalization service for servlets and enterprise beans. By default, the service is disabled for server-side components within the application server. You enable the service by using either the administrative console or the wsadmin tool.
- Enable or disable the internationalization service for EJB clients. By default, the service is disabled within the client container. You enable the service by using the launchClient tool.

Enabling the internationalization service for servlets and enterprise beans

Perform this task to enable the internationalization service in the application server runtime environment.

Any servlet or enterprise bean can use internationalization context if the internationalization service is enabled within the hosting WebSphere Application Server instance.

1. Start the administrative console.
2. Click **Servers > Application servers >** *server_name* **> Container services > Internationalization service**.
3. Enable the internationalization service.
   a. If not already selected, select the **Enable service at server startup** check box.
   b. Click **OK**.

When you select the **Enable service at server startup** setting, the application server automatically initializes and starts the internationalization service whenever the server starts. If you change this setting, be sure to restart the application server for the new setting to take effect.

To disable the service, clear the **Enable service at server startup** check box. In this case, the internationalization service is initialized but not started when the application server starts.

Administration through scripting

Alternatively, the internationalization service can be enabled from the command line by using the wsadmin tool. Start the wsadmin tool and enter the following commands:

```
set x [$AdminConfig list I18NService]
$AdminConfig modify $x { { enable true } }
$AdminConfig save
exit
```

If you enable or disable the internationalization service, be sure to stop and then restart the application server for the new setting to take effect.

Enabling the internationalization service for EJB clients

By default, the internationalization service is disabled for use within Enterprise JavaBeans (EJB) and Web-service enabled client applications. You must enable the service for client applications as well as for all server instances in the runtime environment.

Enable the service.

When calling the launchClient tool, include the argument `-CCDI18NService.enable=true` or `-CCDI18NService.enable=yes`.

Internationalization service settings

Use this page to enable or disable the internationalization service. The internationalization service manages the implicit propagation and scoping of locale and time zone information, called *internationalization context*, within application components. When the service is enabled, application components can use the internationalization context API to programmatically manage locale and time zone information. In turn, components can use that locale and time zone information with the Java 2 Platform,

Standard Edition (J2SE) Internationalization API to perform localizations. If internationalization support is not required on the server, disabling the service can improve performance.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Container services > Internationalization service**.

Enable service at server startup:

Specifies whether the server attempts to start the internationalization service.

Default	Cleared
Range	Valid values are Selected or Cleared

More information about valid values follows:

Selected
When the application server starts, it attempts to start the internationalization service automatically.

Cleared
The server does not try to start the internationalization service.

To enable the internationalization service for applications on this server, the system administrator must select this property and then restart the server.

Internationalization service errors

Certain conditions might cause the internationalization service not to start, to issue java.lang.IllegalStateException exceptions while an application is running, or to exercise default behaviors.

The java.lang.IllegalStateException exception indicates one of the following things:

- An application component attempted an operation that is not supported by the internationalization programming model.

  The IllegalStateException exception is issued whenever a server application component whose internationalization type is set to container-managed internationalization (CMI) attempts to set invocation context. This behavior is a violation of the CMI policy, under which servlets and enterprise beans cannot modify their invocation internationalization context.
- An anomaly occurred that disabled the service.

  For instance, if the internationalization service is not properly initialized, the Java Naming and Directory Interface (JNDI) lookup on the UserInternationalization URL attribute issues a javax.naming.NameNotFoundException exception that contains an IllegalStateException instance.

The following conditions can occur while your internationalized application is running. These conditions might cause the internationalization service not to start, to issue IllegalStateException exceptions, or to exercise default behaviors:

- “The service is disabled ” on page 2034
- “The service is not started” on page 2034
- “Invalid context element” on page 2035
- “Missing context element” on page 2035
- “Invalid policy” on page 2035
- “Missing policy” on page 2035

If you encounter unexpected or exceptional behavior, the problem is likely related to one of these conditions. You need to examine the trace log to investigate these conditions, which requires that you configure the diagnostic trace service to generate messages about internationalization service function. To get started with logging and tracing, see Tracing and logging configuration.

The trace strings for the internationalization service follow; use both:

```
com.ibm.ws.i18n.context.*=all=enabled:com.ibm.websphere.i18n.context.*=all=enabled
```

The service is disabled

The internationalization service is not initialized when the startup setting is cleared. The service generates a message that indicates whether it is enabled or disabled. Applications cannot access the internationalization API when the service is disabled. If an application attempts a JNDI lookup to obtain the UserInternationalization reference, the lookup fails with a NamingException exception, indicating that the reference cannot be found. In addition, the service does not scope (propagate) internationalization context on incoming (outgoing) business method calls.

The service is not started

The internationalization service is operational whenever it is in the STARTED state. For example, if an application attempts to access internationalization context and the service is not started, the API issues an IllegalStateException exception. In addition, the service does not provide runtime support for servlets and enterprise beans.

As an application server progresses through its life cycle, it initializes, starts, stops, and terminates (destroys) the internationalization service. If an anomaly occurs during initialization, the service does not start. After the service is started, its state can change to BLOCKED in the event that a serious error occurs. The service generates a message for every state change.

If a trace message indicates that the service is not STARTED, examine previous messages to determine the problem. For instance, the internationalization service does not start if the activity service is unavailable and a message is displayed to that effect during initialization of the internationalization service.

During startup, the following messages indicate potential configuration or runtime problems:

No ORB support
: The service cannot obtain an instance of the object request broker (ORB). This condition is a fatal error. Examine the SystemErr.log and SystemOut.log files for information.

No TCM support
: The service cannot obtain an instance of its thread context manager (TCM). This condition is a fatal error. Examine the SystemErr.log and SystemOut.log files for information.

No IIOP (activity service) support
: The service cannot register with the activity service. This condition is a fatal error. The internationalization service cannot propagate or receive context on Internet Inter-ORB Protocol (IIOP) requests without activity service support. Examine the SystemErr.log and SystemOut.log files for information.

No AsynchBeans support
: The service cannot register into the asynchronous beans environment. This warning indicates that the asynchronous beans environment cannot support internationalization context.

No EJB container support
: The service cannot register with the Enterprise JavaBeans (EJB) container. This warning indicates that the internationalization service cannot support enterprise beans. Without EJB container support, internationalization contexts do not scope properly to EJB business methods. Review the trace log for any EJB container-related error conditions.

No Web container support
: The service cannot register with the Web container. This warning indicates that the internationalization service cannot support servlets and JavaServer Page (JSP) files. Without Web container support, internationalization contexts do not scope properly to servlet service methods. Review the trace log for any Web container-related error conditions.

No Metadata support
: The service cannot register with the metadata service. This warning indicates that the internationalization service cannot process the internationalization policies within application deployment descriptors. Without metadata support, the service associates the default

internationalization context management policy, [CMI, RunAsCaller], to every servlet lifecycle method and enterprise bean business method invocation. Review the trace log for any metadata service-related error conditions.

No JNDI (Naming service) support

The service cannot bind the UserInternationalization object into the namespace. This condition is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements. Review the trace log for any Naming (JNDI) service-related error conditions.

No API support

The service cannot obtain an instance of an internationalization context API object. This condition is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements.

Invalid context element

The service detected an invalid internationalization context element. For example, the internationalization service does not support TimeZone instances of a type other than java.util.SimpleTimeZone. If the service encounters an unusable element, it logs a message and substitutes the corresponding default element of the JVM.

Missing context element

The service detected a missing internationalization context element. Incoming requests (for example, from application servers that do not support the internationalization service) lack internationalization context. When the service attempts to access a caller internationalization context element (which does not exist in this case), the service logs a message and substitutes the corresponding default element of the Java virtual machine (JVM).

Whenever possible, enable the internationalization service within all clients and hosting application servers that comprise an internationalized enterprise application. Read more information about Administering the internationalization service in the *Administering applications and their environment* PDF book.

Invalid policy

The internationalization service detected a malformed internationalization policy in the application deployment descriptor. The service replaces the malformed attribute with the appropriate default. For instance, if the internationalization type for an entity bean is set to `Application` during the run of a servlet or EJB business method call, the service logs the inconsistency and enforces the `Container` setting instead.

Also, AMI application components do have an implicit container internationalization attribute. By default they run as server. The service silently enforces the implicit policy, `[AMI, RunAsServer]`, and logs messages to this effect.

Invalid container internationalization attributes are likely to occur when specifying the Locales and Time zone ID fields. When encountering invalid locales and time zone IDs within attributes, the service replaces each value with the corresponding default element of the JVM. Be sure to follow the guidelines provided in the *Developing and deploying applications* PDF book.

Missing policy

The service detected a missing internationalization policy. The service replaces the missing policy with the appropriate default. For instance, if the internationalization type is missing for a servlet or enterprise bean, the service sets the attribute to Container.

Container internationalization attributes are not mandatory for CMI application components. In the event that a CMI servlet or EJB business method lacks a container internationalization attribute, the service silently enforces the implicit policy `[CMI, RunAsCaller]`.

When an application lacks internationalization policies in its deployment descriptor, or metadata support is unavailable, the service logs a message and applies the policy `[CMI, RunAsCaller]` on every servlet service method and EJB business method invocation.

Read the information in the *Developing and deploying applications* PDF book:
- Assembling internationalized applications
- Container internationalization attributes
- Internationalization type

Object pools

Using object pools

An object pool helps an application avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused.

Object pools are not meant to be used for pooling JDBC connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To use an object pool, the product administrator must define an *object pool manager* using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Note: The Object pool manager service is only supported from within the EJB container or Web container. Looking up and using a configured object pool manager from a Java 2 Platform Enterprise Edition (J2EE) application client container is not supported.

1. Start the administrative console.
2. Click **Resources > Object pool managers**.
3. Specify a **Scope** value and click **New**.
4. Specify the required properties for work manager settings.
   Scope The scope of the configured resource. This value indicates the location for the configuration file.
   Name The name of the object pool manager. This name can be up to 30 ASCII characters long.
   JNDI Name
   The Java Naming and Directory Interface (JNDI) name for the pool manager.
5. [Optional] Specify a **Description** and a **Category** for the object pool manager.

After you have completed these steps, applications can find the object pool manager by doing a JNDI lookup using the specified JNDI name.

The following code illustrates how an application can find an object pool manager object:

```
InitialContext ic = new InitialContext();
ObjectPoolManager opm = (ObjectPoolManager)ic.lookup("java:comp/env/pool");
```

When the application has an ObjectPoolManager, it can cache an object pool for classes of the types it wants to use. The following is an example:

```
ObjectPool arrayListPool = null;
ObjectPool vectorPool = null;
try
{
```

```
 arrayListPool = opm.getPool(ArrayList.class);
 vectorPool = opm.getPool(Vector.class);
}
catch(InstantiationException e)
{
 // problem creating pool
}
catch(IllegalAccessException e)
{
 // problem creating pool
}
```

When the application has the pools, the application can use them as in the following example:

```
ArrayList list = null;
try
{
 list = (ArrayList)arrayListPool.getObject();
 list.clear(); // just in case
 for(int i = 0; i < 10; ++i)
 {
  list.add("" + i);
 }
 // do what ever we need with the ArrayList
}
finally
{
 if(list != null) arrayListPool.returnObject(list);
}
```

This example presents the basic pattern for using object pooling. If the application does not return the object, then the only adverse effect is that the object cannot be reused.

Object pool managers

Object pool managers control the reuse of application objects and Developer Kit objects, such as Vectors and HashMaps.

Multiple object pool managers can be created in an Application Server cell. Each object pool manager has a unique cell-wide Java Naming and Directory Interface (JNDI) name. Applications can find a specific object pool manager by doing a JNDI lookup using the specific JNDI name.

The object pool manager and its associated objects implement the following interfaces:

```
public interface ObjectPoolManager
{
 ObjectPool getPool(Class aClass)
  throws InstantiationException, IllegalAccessException;
 ObjectPool createFastPool(Class aClass)
  throws InstantiationException, IllegalAccessException;

}

public interface ObjectPool
{
 Object getObject();
 void returnObject(Object o);

}
```

Each object pool manager can be used to pool any Java object with the following characteristics:

- The object must be a public class with a public default constructor.
- If the object implements the java.util.Collection interface, it must support the optional clear() method.

Each pooled object class must have its own object pool. In addition, an application gets an object pool for a specific object using either the ObjectPoolManager.getPool() method or the ObjectPoolManager.createFastPool() method. The difference between these methods is that the getPool() method returns a pool that can be shared across multiple threads. The createFastPool() method returns a pool that can only be used by a single thread.

If in a Java virtual machine (JVM), the getPool() method is called multiple times for a single class, the same pool is returned. A new pool is returned for each call when the createFastPool() method is called. Basically, the getPool() method returns a pool that is thread-synchronized.

The pool for use by multiple threads is slightly slower than a fast pool because of the need to handle thread synchronization. However, extreme care must be taken when using a fast pool. Consider the following interface:

```
public interface PoolableObject
{
 void init();
 void returned();
}
```

If the objects placed in the pool implement this interface and the ObjectPool.getObject() method is called, the object that the pool distributes has the init() method called on it. When the ObjectPool.returnObject() method is called, the PoolableObject.returned() method is called on the object before it is returned to the object pool. Using this method objects can be pre-initialized or cleaned up.

It is not always possible for an object to implement PoolableObject. For example, an application might want to pool ArrayList objects. The ArrayList object needs clearing each time the application reuses it. The application might extend the ArrayList object and have the ArrayList object implement a poolable object. For example, consider the following:

```
public class PooledArrayList extends ArrayList implements PoolableObject
{
 public PooledArrayList()
 {
 }

 public void init() {
 }

 public void returned()
 {
  clear();
 }
}
```

If the application uses this object, in place of a true ArrayList object, the ArrayList object is cleared automatically when it is returned to the pool.

Clearing an ArrayList object simply marks it as empty and the array backing the ArrayList object is not freed. Therefore, as the application reuses the ArrayList, the backing array expands until it is big enough for all of the application requirements. When this point is reached, the application stops allocating and copying new backing arrays and achieves the best performance.

It might not be possible or desirable to use the previous procedure. An alternative is to implement a custom object pool and register this pool with the object pool manager as the pool to use for classes of that type. The class is registered by the WebSphere administrator when the object pool manager is defined in the cell. Take care that these classes are packaged in Java Archive (JAR) files available on all of the nodes in the cell where they might be used.

Object pool managers collection

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers**.

To use an object pool, the product administrator must define an object pool manager using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Name:

Specifies the name by which the object pool manager is known for administrative purposes.

Data type	String
Range	1 through 30 ASCII characters

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the object pool manager.

Data type	String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Description:

Specifies the description of the object pool manager.

Data type	String

Category:

Specifies the category name used to classify or group this object pool manager.

Data type	String

Object pool managers settings:

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers >** *objectpoolmanager_name*

To use an object pool, the product administrator must define an object pool manager using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

The name by which the object pool manager is known for administrative purposes.

Data type	String
Range	1 through 30 ASCII characters

JNDI Name:

The Java Naming and Directory Interface (JNDI) name for the object pool manager.

Data type	String

Description:

A description of the object pool manager.

Data type	String

Category:

A category name used to classify or to group this object pool manager.

Data type	String

Custom object pool collection:

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers >** *objectpoolmanager_name* **> Custom object pools**.

Use custom object pools to insert additional logic around the following mechanisms:

- Constructing an object pool (A list of properties can be set)
- Flushing the object pool
- Getting objects from the pool
- Returning objects from the pool

These features allow for actions such as, clearing the state of an object when returning it to the pool, configuring the state of an object when retrieving it from the pool, or configuring generic pools and sending instructions on how to behave using custom properties.

To use an object pool the product administrator must define an object pool manager using the administrative console. You can create multiple object pool managers in an Application Server cell.

Pool class name:

Specifies the fully qualified class name of the objects that are stored in the custom object pool.

Data type String

Pool implementation class name:

Specifies the fully qualified class name of the implementation class for the custom object pool.

Data type String

Custom object pool settings:

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers >** *objectpoolmanager_name* **> Custom object pools >** *objectpool_name*.

Use custom object pools to insert additional logic around the following mechanisms:

- Constructing an object pool (A list of properties can be set)
- Flushing the object pool
- Getting objects from the pool
- Returning objects from the pool

These features allow for actions such as, clearing the state of an object when returning it to the pool, configuring the state of an object when retrieving it from the pool, or configuring generic pools and sending instructions on how to behave using custom properties.

To use an object pool, the product administrator must define an object pool manager using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Pool Class Name:

The fully qualified class name of the objects that are stored in the object pool.

Data type String

Pool Impl Class Name:

The fully qualified class name of the CustomObjectPool implementation class for this object pool.

Data type String

Object pool service settings
Use this page to enable or disable the object pool service, which manages object pool resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* > Container services > **Object Pool Service**.

Enable service at server startup:

Specifies whether the server attempts to start the object pool service.

Default
Cleared

Range

Selected
When the application server starts, it attempts to start the object pool service automatically.

Cleared
The server does not try to start the object pool service. If object pool resources are used on this server, then the system administrator must start the object pool service manually or select this property, and then restart the server.

Object pools: Resources for learning
This topic provides links to find relevant supplemental information about object pools.

Use the following links to find relevant supplemental information about object pools. 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.

Furthermore, these links provide guidance on using object pools. Since object pooling is a general topic and the WebSphere Application Server product implementation is only one way to use it, you must understand when object pooling is necessary. These articles help you make that decision.

Programming model and decisions
- Build your own ObjectPool in Java to boost application speed
- Improve the robustness and performance of your ObjectPool
- Recycle broken objects in resource pools

MBeans for object pool managers and object pools

Legacy MBean names for object pool managers and object pools are deprecated. The legacy names are based on the object pool manager name (which is not required to be unique) rather than the object pool manager JNDI name. For object pools, the legacy name is also lacking any identifier of the version of the pooled class. Additionally, object pool Performance Monitoring Instrumentation (PMI) statistics are aggregated for object pools with the same legacy object pool MBean name.

For example, if the object pool manager and pooled class are as follows:

```
object pool manager name:              My ObjectPool
object pool manager JNDI name:         op/MyObjectPool
pooled class name:                     java.util.ArrayList
hash code of java.util.ArrayList.class: 1111eb3f (hexadecimal)
```

the legacy object pool manager MBean name will be:

```
ObjectPoolManager_My ObjectPool
```

and the legacy object pool MBean name will be:

```
ObjectPool_My ObjectPool_java.util.ArrayList
```

Instead of using the deprecated legacy MBean names, use the MBean names that are based on the JNDI name of the object pool manager.

For the example above, the JNDI name-based object pool manager MBean name is:

```
ObjectPoolManager_op/MyObjectPool
```

and the JNDI name-based object pool MBean name is:

```
ObjectPool_op/MyObjectPool_java.util.ArrayList.class@1111eb3f
```

Formats for MBean names

Type	Name format
Deprecated legacy object pool manager MBean name:	ObjectPoolManager_[object pool manager name]
JNDI name-based object pool manager MBean name:	ObjectPoolManager_[object pool manager JNDI name]
Deprecated legacy object pool MBean name:	ObjectPool_[object pool manager name]_[pooled class name]
JNDI name-based object pool MBean name:	ObjectPool_[object pool manager JNDI name]_[pooled class name].class@[hexadecimal representation of the hash code of the pooled class' java.lang.Class reference]

In all of the above formats, characters that are not valid for MBean names are replaced with the '.' character.

MBeans for object pool managers and object pools

Legacy MBean names for object pool managers and object pools are deprecated. The legacy names are based on the object pool manager name (which is not required to be unique) rather than the object pool manager JNDI name. For object pools, the legacy name is also lacking any identifier of the version of the pooled class. Additionally, object pool Performance Monitoring Instrumentation (PMI) statistics are aggregated for object pools with the same legacy object pool MBean name.

For example, if the object pool manager and pooled class are as follows:

```
object pool manager name:               My ObjectPool
object pool manager JNDI name:          op/MyObjectPool
pooled class name:                      java.util.ArrayList
hash code of java.util.ArrayList.class: 1111eb3f (hexadecimal)
```

the legacy object pool manager MBean name will be:

```
ObjectPoolManager_My ObjectPool
```

and the legacy object pool MBean name will be:

```
ObjectPool_My ObjectPool_java.util.ArrayList
```

Instead of using the deprecated legacy MBean names, use the MBean names that are based on the JNDI name of the object pool manager.

For the example above, the JNDI name-based object pool manager MBean name is:

```
ObjectPoolManager_op/MyObjectPool
```

and the JNDI name-based object pool MBean name is:

```
ObjectPool_op/MyObjectPool_java.util.ArrayList.class@1111eb3f
```

Formats for MBean names

Type	Name format
Deprecated legacy object pool manager MBean name:	ObjectPoolManager_[object pool manager name]
JNDI name-based object pool manager MBean name:	ObjectPoolManager_[object pool manager JNDI name]
Deprecated legacy object pool MBean name:	ObjectPool_[object pool manager name]_[pooled class name]
JNDI name-based object pool MBean name:	ObjectPool_[object pool manager JNDI name]_[pooled class name].class@[hexadecimal representation of the hash code of the pooled class' java.lang.Class reference]

In all of the above formats, characters that are not valid for MBean names are replaced with the '.' character.

Scheduler

Using schedulers

Schedulers enable J2EE application tasks to run at a requested time. Schedulers also enable application developers to create their own stateless session EJB components to receive event notifications during a task life cycle, allowing the plugging-in of custom logging utilities or workflow applications.

You can schedule the following types of tasks:
- Invoke a session bean method
- Send a Java Message Service (JMS) message to a queue or topic

Stateless session EJB components are also used to provide generic calendaring. Developers can either use the supplied calendar bean or create their own for their existing business calendars. For example, one of your business processes might involve invoicing for services. With the scheduler's use of stateless EJB components, you can schedule when periodic email distributions are to be sent to your customers who have received invoices. The scheduler service performs these tasks, repeating as necessary, according to the metadata for that task.

A scheduler is the mechanism by which the timer service for Enterprise Java Beans 2.1 runs. You can configure the EJB timer service to use many of the features that schedulers provide. See the timer service for Enterprise Java Beans 2.1 documentation for more details.

Use the following table to determine which persistent timer service is best for you:

Schedulers	EJB timers
Run stateless session EJB components and sends JMS messages	Run all EJB types except for stateful session beans
Persistent, transactional and highly available.	Persistent, transactional and highly available.
Tasks guaranteed to run only once	Timers guaranteed to run only once, if the timer EJB uses a container-managed global transaction
Run repeating tasks using any calculation rules	Run repeating tasks using a repeating interval defined in milliseconds

Uses a modified fixed-delay time calculation to determine repeating intervals (next run time based on the start-time of the previous task)	Uses a fixed-rate time calculation to determine repeating intervals (time of the next task is based on the original scheduled time).
Programmatic task monitoring capability with the use of the NotificationSink stateless session EJB	No programmatic timer monitoring
Abort late or time-sensitive tasks from running	Abort late or time-sensitive tasks from running (achieved through manual detection within the javax.ejb.TimedObject implementation).
Manage any task lifecycle (find, suspend, resume, cancel and purge tasks programmatically and through Java Management Extensions (JMX)).	Find and cancel its timers programmatically. Administrators find and cancel timers using a command-line utility.
Store a limited amount of text with the data, like a **Name** (arbitrary data stored externally.)	Store arbitrary data with a timer

This task demonstrates how to manage, develop and interoperate with schedulers and subsequent tasks.

1. Manage the scheduler service. This article includes instructions for creating and configuring schedulers, creating and configuring a database for schedulers and administering schedulers.
2. Develop and schedule tasks. This article includes instructions for developing various types of tasks, receiving notifications from a task, submitting tasks to a scheduler, and managing tasks.

   Note: Creating and manipulating scheduled tasks through the Scheduler API interface is only supported from within the Enterprise Java Beans (EJB) container or Web container (JavaServer Pages or servlets). Looking up and using a configured scheduler from a Java 2 Platform Enterprise Edition (J2EE) application client container is not supported.
3. Interoperate with schedulers. This article explains how to manage scheduler in a clustered environment with mixed WebSphere Application Server product versions and mixed platforms.

Scheduler daemon

A scheduler daemon is a background thread that searches for tasks to run in the database.

A scheduler daemon is started for each scheduler defined on each server. If `Scheduler 1` is configured on `server1`, then only one scheduler daemon runs on `server1` unless it is cloned. If `Scheduler 1` is defined at the node scope level, then the scheduler will run on each server within that node.

The poll interval determines the frequency at which the persistent store is queried. By default, this value is set to 30 seconds. When a task is found that is scheduled to run within the current poll interval, an asynchronous beans alarm is set. The task then runs as close to this time as possible using an alarm thread from the scheduler's associated work manager. Thus, the number of alarm threads configured on the work manager determines how many concurrent tasks are executed. No tasks are lost. If we reach this limit, then new tasks are simply queued to be executed when an alarm thread becomes available. The actual firing time is dictated by server load and availability of free threads in the alarm thread pool of the associated work manager.

Scheduler daemons in a cluster

When multiple schedulers are configured to use the same tables (as is the case in a clustered environment), any of the daemons can find a task and set the alarm in its Java virtual machine (JVM). The task is executed in the virtual machine where the scheduler daemon first runs, until the daemon is stopped and another daemon starts. If an application on `server1` schedules a task to run and `server2` was started before `server1`, then the task runs on `server2`.

Example: Stopping and starting scheduler daemons using Java Management Extensions API:

Use the wsadmin scripting tool to invoke a `Jacl` script and stop and start a scheduler daemon.

This example JACL script can be invoked using the wsadmin scripting tool. It will attempt to stop and start a scheduler daemon.

```
# Example JACL Script to restart a Scheduler Daemon

set schedJNDIName sched/MyScheduler

# Find the WASScheduler MBean
regsub -all {/} $schedJNDIName "." schedJNDIName
set mbeanName Scheduler_$schedJNDIName
puts "Looking up Scheduler MBean $mbeanName"
set sched [$AdminControl queryNames WebSphere:*,type=WASScheduler,name=$mbeanName]

# Invoke the stopDaemon operation.
puts "Stopping the daemon..."
$AdminControl invoke $sched stopDaemon
puts "The daemon has stopped."

# Invoke the startDaemon operation.
puts "Starting the daemon..."
$AdminControl invoke $sched startDaemon 0
puts "The daemon has started."
```

Example: Dynamically changing scheduler daemon poll intervals using Java Management Extensions API:

Use the wsadmin scripting tool to invoke a `Jacl` script and dynamically change scheduler daemon poll intervals.

To dynamically change scheduler daemon poll intervals, use the wsadmin scripting tool to invoke this example `JACL` script. Invoking this example sets the poll interval of the scheduler daemon to 60 seconds.

```
# Example JACL Script to set the Scheduler daemon's poll interval

set schedJNDIName sched/MyScheduler

# Find the WASScheduler MBean
regsub -all {/} $schedJNDIName "." schedJNDIName
set mbeanName Scheduler_$schedJNDIName
puts "Looking-up Scheduler MBean $mbeanName"
set sched [$AdminControl queryNames WebSphere:*,type=WASScheduler,name=$mbeanName]

# Set the poll interval to 60 seconds (60000 ms)
$AdminControl setAttribute $sched pollInterval 60000
puts "Poll interval set."
```

Interoperating with schedulers

Schedulers support forward compatibility. Tasks created in previous versions of WebSphere Application Server Enterprise Edition 5.0 or WebSphere Business Integration Server Foundation 5.1 continue to run in WebSphere Application Server, Version 6.x schedulers. Tasks that you create using Version 6.x are not compatible with product schedulers from Version 5.x. Version 5.x schedulers do not run any Version 6.x tasks.

Schedulers and versions

All schedulers that are configured to use the same database and tables are considered a clustered scheduler. To guarantee that your tasks run correctly, all servers in a scheduler cluster must be at the same version. If the servers are at different versions, tasks created with a Version 6.x scheduler might not run. If a mixed-Version environment is required for a short period of time, then all scheduler poll daemons should be stopped on all Version 5.x servers to allow a Version 6.x server to run all tasks. This action allows the Version 6.x schedulers to obtain leases and run tasks that have been created with a Version 6.x scheduler.

Running tasks created with schedulers prior to Version 5.0.2 is not supported. See the topic, ″Interoperating with the Scheduler service,″ in the WebSphere Application Server Enterprise Edition Version 5.0.2 information center for details on how to migrate these tasks to a more recent version. See the Information Center Library to access the Version 5.0.2 information center.

Scheduler calendars

The scheduler provides stateless session bean interfaces which allow creating common calendars which can be used by the scheduler and any J2EE application.

The `SchedulerCalendars.ear` application is available and provides a default UserCalendar EJB implementation which allows using the SIMPLE and CRON calendars. Although this application is not required when using the scheduler, it is available to use from any J2EE application.

For details on how the SIMPLE and CRON calendars behave, see the API documentation for the com.ibm.websphere.scheduler.UserCalendar interface.

Specifying a UserCalendar with the scheduler

A UserCalendar is specified using the setUserCalendar() method of the TaskInfo interface of the scheduler. This interface allows you to select the JNDI name of the home interface of a UserCalendar bean. Because some UserCalendar bean implementations might handle multiple types of calendars, the interface also allows you to optionally select which type of calendar to use. A list of valid calendar types can be retrieved by invoking the getCalendarNames() method of the UserCalendar interface.

If the setUserCalendar() method is not invoked, or if a value of null or empty-string is specified for the home JNDI name parameter, then the default UserCalendar is used internally by the scheduler. When the default UserCalendar is accessed internally, it is not necessary that the `SchedulerCalendars.ear` system application be installed.

You might want to use the default UserCalendar directly in your other J2EE applications, apart from the scheduler. In this case, you may use the UserCalendarHome.DEFAULT_CALENDAR_JNDI_NAME value to look up the default UserCalendar from your applications. You may also supply this value to the setUserCalendar() method of the TaskInfo interface. You will need to ensure the `SchedulerCalendars.ear` system application was either automatically installed or that you have installed it manually.

Scheduler service settings

Use this page to enable or disable the scheduler service. The scheduler service manages scheduler resources used by the server. The administrative console page used to configure the scheduler service is not available for version 6 (and above) servers. It is only available for version 5.*x* servers.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Scheduler Service**.

Startup:

Specifies whether the server attempts to start the scheduler service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the scheduler service automatically.

Cleared
The server does not try to start the scheduler service. If scheduler resources are to be used on this server, the system administrator must start the scheduler service manually or select this property, then restart the server.

Managing schedulers

Schedulers are configured using the administrative console, configuration service or scripting and are available to all servers on which a scheduler is visible.

You can create multiple schedulers within a single server, cluster, node or cell. Each configured scheduler is an independent task scheduling engine that has a unique Java Naming and Directory Interface (JNDI) name, persistent storage device and daemon.

1. Configure schedulers.
2. Create the database for schedulers.

Configuring schedulers

Before your application can make use of the scheduler service, you must configure a scheduler using the administrative console, configuration service or scripting. Conceptually, a scheduler is similar to a data source in that you must specify various configuration attributes, including a JNDI name where the instance is bound. Once defined, an application using the Scheduler API or WASScheduler MBean can look up the scheduler object and call various methods to manage tasks.

The scheduler service is always enabled. In previous versions of the product, the scheduler service could be disabled using the administrative console or configuration service. Scheduler service configuration objects are present in the configuration service, but the enabled attribute is ignored.

To achieve high availability, you can configure a duplicate scheduler on each server in a cluster, or create a scheduler at the cluster scope. For example, each server that contains a scheduler with the JNDI name `sched/MyScheduler`, with the same database configuration parameters (data source and table prefix) behaves as a single clustered scheduler. Each server in the scheduler cluster has a running scheduler instance, which increases the number of poll daemons and allows automatic failover. For more information on creating clusters for high availability, see the article, ″WebSphere Enterprise Scheduler planning and administration guide.″

Typically, create schedulers at the server or cluster scope. Scheduler poll daemons run in each server within the configured scope, which means that if you create a scheduler at the node or cell scope, the scheduler poll daemon can attempt to run tasks on any of the servers in the node or cell. If applications are not mapped uniformly over each server in that scope, the scheduler might not run tasks correctly. Since applications are mapped to servers and clusters, there is less chance for error and less competition between daemons to run tasks.

Depending on your preferred method of configuration, select one of the following steps to configure schedulers.

1. Configuring schedulers using the administrative console.
2. Configuring schedulers using Java Management Extensions API (JMX).

A scheduler is configured and ready to use.

Configuring scheduler default transaction isolation:

The scheduler uses read-committed transaction isolation, by default, when reading tasks using the get or find APIs on the com.ibm.websphere.scheduler.Scheduler interface and WASScheduler MBean. The default behavior for a scheduler can be changed to read-uncommitted, which allows the get and find methods to return the current or next state of the task in the database. This topic describes how to change the default behavior for the get and find methods.

See the scheduler API documentation to view the com.ibm.websphere.scheduler.TaskInfo.setTaskExecutionOptions() method, which details how to return the next state of the task or the current state of the task.

Note: If the scheduler database does not support uncommitted reads, such as Oracle, this parameter has no effect.

To change the default behavior for the get and find methods, complete the following steps:

1. From the administrative console, click **Resources** > **Schedulers** > *scheduler_name*.
2. Click **Custom Properties**.
3. Click **New**.
4. Add the following properties:

Name	defaultReadTransactionIso
Type	java.lang.Integer
Value	`1 (for read-uncommitted transaction isolation)` `2 (for read-committed transaction isolation)`

5. Click **Apply**.
6. Click **OK**.
7. Save the changes, and verify that you initiate a file synchronization before restarting the servers.
8. Restart the Application Server for the changes to take effect.

Configuring schedulers using the administrative console:

Schedulers can be created or configured using the administrative console.

1. Start the administrative console.
2. Select **Resources** > **Schedulers**.
3. Click **New**.
4. Specify configuration settings. Fields marked with an asterisk (*) are required. The settings are described in detail in the topic ″Scheduler settings.″
5. Click OK or Apply to save the changes.
6. Save the changes to the configuration repository.

A scheduler is now configured and ready to use for newly installed applications. If the scheduler JNDI name is not yet visible to your application, restarting the application or restarting application server will allow the scheduler to be seen.

When schedulers are created for the first time, the poll daemon will not automatically start and must be started manually and will only start automatically the next time the server is started. To start the poll daemon manually, see the scheduler daemons topic.

Note: Changes to existing scheduler configurations will not take affect until after the application server is restarted.

Schedulers collection:

Use this page to manage scheduler configurations. Schedulers are persistent and transactional timer services that can run business logic. Each scheduler runs tasks independently and has a programming interface accessible from J2EE applications using the Java Naming and Directory Interface (JNDI). You can also manage schedulers using a Java Management Extensions (JMX) MBean. See the scheduler documentation in the Information Center for details on how to configure and use schedulers.

To view this administrative console page, click **Resources > Schedulers**.

Name:

Specifies the name of the data source where persistent tasks are stored.

Data type String

JNDI name:

Specifies the JNDI name of the work manager, which is used to manage the number of tasks that can run concurrently with the scheduler. The work manager also can limit the amount of J2EE context applied to the task.

The JNDI name specifies where this scheduler instance is bound in the name space. Clients can look this name up directly, although the use of resource references is recommended.

Data type String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Data source JNDI name:

Specifies the alias for the user name and password that are used to access the data source.

Any data source available in the name space can be used with a scheduler. Multiple schedulers can share a single data source while using different tables by specifying a table prefix.

Data type String

Table prefix:

Specifies the string prefix to affix to the scheduler tables.

Multiple independent schedulers can share the same database if each instance specifies a different prefix string.

Data type String

Poll interval:

Specifies the interval, in seconds, that a scheduler polls the database. The default value is appropriate for most applications.

Each poll operation can be consuming. If the interval is extremely small and there are many scheduled tasks, polling can consume a large portion of system resources.

Data type Integer
Units Seconds
Default 30
Range Any positive long integer

Work manager JNDI name:

Specifies the JNDI name of the work manager, which is used to manage the number of tasks that can run concurrently with the scheduler. The work manager also can limit the amount of J2EE context applied to the task.

The work manager is a server object that serves as a logical thread pool for the scheduler. Each repeating task that is created using this scheduler uses the **Number of alarm threads** specified in the work manager, which affects the number of tasks that can run concurrently. Use the work manager **Service Names** property to limit the amount of context information that is propagated to the task when it runs.

When a task runs, the task is run in the work manager associated with the scheduler instance. You can control the number of actively running tasks at a given time by configuring schedulers with a specific work manager. The number of tasks that can run concurrently is governed by the **Number of alarm threads** parameter on the work manager.

Note: When you configure a scheduler resource on a Version 5.x node or server, the scheduler must reference a work manager in the same scope. For example, a scheduler instance configured at the server1 scope must use a work manager also configured at the server1 scope.

Verify tables:

Specifies to validate that scheduler data sources, table prefixes, security authentication information and tables are configured correctly.

You can use this verification method in production and development environments without altering database properties.

Create tables:

Specifies to create the necessary tables and indices required for a scheduler to operate.

This method of creating scheduler tables is designed for simple topologies and development environments. Use the supplied scheduler data definition language files for advanced or production environments and for databases that do not support this feature. .

Drop tables:

Specifies the removal of tables and indices required for schedulers to operate.

This method of removing scheduler tables and indices is recommended for development environments and does not delete previously scheduled tasks.

Schedulers settings:

Use this page to modify scheduler settings.

To view this administrative console page, click **Resources > Schedulers >** *scheduler_name*.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

Specifies the name by which this scheduler is known for administrative purposes.

Data type String

JNDI name:

Specifies the name of the data source where persistent tasks are stored.

The JNDI name specifies where this scheduler instance is bound in the namespace. Clients can look this name up directly, although the use of resource references is recommended.

Data type String

Description:

Specifies the description of this scheduler for administrative purposes.

Data type String

Category:

Specifies a string that can be used to classify or group this scheduler.

Data type String

Data source JNDI name:

Specifies the name of the data source where persistent tasks are stored.

Any data source available in the name space can be used with a scheduler. Multiple schedulers can share a single data source while using different tables by specifying a table prefix.

Data type String

Data source alias:

Specifies the alias for the user name and password that are used to access the data source.

Data type String

Table prefix:

Specifies the string prefix to affix to the scheduler tables. Multiple independent schedulers can share the same database if each scheduler specifies a different prefix string.

Multiple independent schedulers can share the same database if each instance specifies a different prefix string.

Note: Use a table prefix with all capital characters. If lowercase characters are used for the table prefix, they are automatically capitalized at run time.

Data type String

Poll interval:

Specifies the interval, in seconds, that a scheduler polls the database. The default value is appropriate for most applications.

Each poll operation can be consuming. If the interval is extremely small and there are many scheduled tasks, polling can consume a large portion of system resources.

Data type	Integer
Units	Seconds
Default	30
Range	Any positive long integer

Work manager JNDI name:

Specifies the JNDI name of the work manager, which is used to manage the number of tasks that can run concurrently with the scheduler. The work manager also can limit the amount of J2EE context applied to the task.

The work manager is a server object that serves as a logical thread pool for the scheduler. Each repeating task that is created using this scheduler uses the **Number of alarm threads** specified in the work manager, which affects the number of tasks that can run concurrently. Use the work manager **Service Names** property to limit the amount of context information that is propagated to the task when it runs.

When a task runs, the task is run in the work manager associated with the scheduler instance. You can control the number of actively running tasks at a given time by configuring schedulers with a specific work manager. The number of tasks that can run concurrently is governed by the **Number of alarm threads** parameter on the work manager.

Note: When you configure a scheduler resource on a Version 5.x node or server, the scheduler must reference a work manager in the same scope. For example, a scheduler instance configured at the server1 scope must use a work manager also configured at the server1 scope.

Use administration roles:

Specifies that when this option and administrative security are both enabled, the user administration roles are enforced when the scheduler JMX commands or APIs are used to create and modify tasks. If this option is not enabled, all the users can create and modify tasks.

Schedulers require several user roles to plan for, develop, administer and operate the scheduler service: administrator, developer and operator.

Data type	check box
Default	unchecked
Range	• Operator, Administrator --Calls any of the scheduler MBean or API methods and runs any of the scheduler administrative console functions. • Monitor, Configurator --Calls the scheduler MBean or API methods, but cannot create tasks or modify the state of any tasks. Only read-only methods and properties are accepted.

Configuring schedulers using Java Management Extensions:

Schedulers can be created or configured using the Java Management Extensions (JMX) API using one of several scripting languages or Java.

To run with Java, two JAR files need to be present in the program class path: `wsexception.jar` and `wasjmx.jar`.

Complete these steps when using Java programs that utilize JMX.

1. Look up the host and get an administration client handle.
2. Get a configuration service handle.
3. Update the `resource-pme.xml` file using the configuration service, as needed.
   a. Find the SchedulerProvider for a given scope.
   b. Create a SchedulerConfiguration and specify all required parameters identifying the SchedulerProvider as the parent object.
4. Reload the `resource-pme.xml` file to bind the newly created scheduler into the JNDI namespace. Perform this step if you want to use the newly created scheduler immediately, without restarting the application server.
   a. Locate the DataSourceConfigHelper MBean using the name.
   b. Invoke the reload() operation.

A scheduler is now configured and ready to use for newly installed applications. If the scheduler JNDI name is not yet visible to your application, reinstalling the application or restarting the application server will allow the scheduler to be seen.

When schedulers are created for the first time, the poll daemon does not automatically start, and you must start it manually. When you restart the server, the poll daemon starts automatically. To start the poll daemon manually, see scheduler daemons.

Note: Changes to existing scheduler configurations will not take affect until after the application server is restarted.

Example: Using scripting to create and configure schedulers:

Use the wsadmin scripting tool to invoke a `Jacl` script and create a SchedulerConfiguration resource.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which creates a SchedulerConfiguration resource using the DefaultWorkManager at the server scope.

```
# Example JACL Script to create a SchedulerConfiguration
# at the server scope

# Change the cell, node and server to match your environment
set cellName   MyCell
set nodeName   MyNode
set serverName server1


# We can just grab the first provider, since there is only one at the
# server scope level.

set schedProv [$AdminConfig getid /Cell:$cellName/Node:$nodeName/Server:$serverName/
  SchedulerProvider:SchedulerProvider]
if {$schedProv == ""} {
    puts "Unable to find SchedulerProvider for server: $serverName.  Aborting."
    exit
}
puts "Found a SchedulerProvider"

# Create a WorkManager for our scheduler at the server scope.
# We could use any of the other scopes as long as it is at the same
# or higher than the Scheduler's scope.

set wrkMgrProv [$AdminConfig getid /Cell:$cellName/Node:$nodeName/Server:$serverName/
```

```
  WorkManagerProvider:WorkManagerProvider/]
if {$wrkMgrProv == ""} {
    puts "Unable to find the WorkManagerProvider for server: $serverName.  Aborting."
    exit
}
puts "Found a WorkManagerProvider"

set wmName             "MyScheduler WorkManager"
set wmJNDIName         "wm/MySchedWorkManager"
set wmIsGrowable       false
set wmMaxThreads       1
set wmMinThreads       0
set wmNumAlarmThreads  10
set wmServiceNames     "com.ibm.ws.i18n;security;UserWorkArea;zos.wlm"
set wmThreadPriority   5

# Setup our DefaultWorkManager attributes
set createAttrs [subst { \
    {isGrowable $wmIsGrowable} \
    {jndiName $wmJNDIName} \
    {maxThreads $wmMaxThreads} \
    {minThreads $wmMinThreads} \
    {name "$wmName"} \
    {numAlarmThreads $wmNumAlarmThreads} \
    {serviceNames "$wmServiceNames"} \
    {threadPriority $wmThreadPriority} }]

puts "Creating a WorkManager"
$AdminConfig create WorkManagerInfo $wrkMgrProv $createAttrs
puts "WorkManager Created"

# Setup our SchedulerConfiguration attributes
set schedulerName          MyScheduler
set schedulerJNDIName      sched/MyScheduler
set datasourceJNDIName     jdbc/MySchedulerDatasource
set datasourceAlias        MySchedulerAlias
set pollInterval           30
set tablePrefix            MSCD
set useAdminRoles          true

set createAttrs [subst { \
  {name $schedulerName} \
  {datasourceJNDIName $datasourceJNDIName} \
  {datasourceAlias $datasourceAlias} \
  {jndiName $schedulerJNDIName} \
  {pollInterval $pollInterval} \
  {tablePrefix $tablePrefix} \
  {useAdminRoles true} \
  {workManagerInfoJNDIName $wmJNDIName}}]

puts "Creating a Scheduler"
$AdminConfig create SchedulerConfiguration $schedProv $createAttrs
puts "Scheduler created"

# Save the configuration
$AdminConfig save
```

Related tasks

“Configuring schedulers using Java Management Extensions” on page 2053
Schedulers can be created or configured using the Java Management Extensions (JMX) API using one of several scripting languages or Java.

Using scripting (wsadmin)
The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language.

Creating a scheduler resource reference:

When you define schedulers in the server configuration, the object instance is bound into the global name space under the configured Java Naming Directory Interface (JNDI) name. You can use a resource reference to avoid manually coding this JNDI name into your application. Using a resource reference allows administrators to map applications to the appropriate schedulers.

You can alternatively create a scheduler resource reference by editing the XML directly. A Scheduler resource reference is a Java 2 Platform Enterprise Edition (J2EE) compliant resource that uses the class com.ibm.websphere.scheduler.Scheduler as the object type. For information regarding the XML file format, see the J2EE Specification.

1. Start an assembly tool, such as Application Server Toolkit or Rational Application Developer.
2. Open the J2EE perspective.
3. Open your EJB or Web module with the Deployment Descriptor Editor.
4. Click the **Reference** tab at the bottom of the window.
5. Click **Add**.
6. Select the **Resource reference** option.
7. Click **Next**.
8. Complete the Reference fields as shown in the following properties:
   Name The reference name, for example, *sched/MyScheduler*. According to this example, the name you choose has a local reference name of **java:comp/env/*sched/MyScheduler***.
   Type Select **com.ibm.websphere.scheduler.Scheduler**, and click **OK**.
   Authentication
   Select container.
   Description
   Any relevant description.
9. Click finish.
10. (**Optional**) Enter a global JNDI name of a configured scheduler in the JNDI name field in the **Bindings** section of the **Reference** window. You can specify or override this value when you install the application.
11. Save your changes to the deployment descriptor.

A scheduler resource reference is now available to use within your application

Creating the database for schedulers

Each scheduler requires a database in which to store its persistent information. Schedulers use this database for storing tasks and then running them. The choice of database and location should be determined by the application developer and server administrator.

Scheduler performance is ultimately limited by database performance. If you need more tasks per second, you can run the scheduler daemons on larger systems, use clusters for the session beans used by the tasks or partition the tasks by using multiple schedulers. Eventually, however, the scheduler database becomes saturated, and a larger or better-tuned database system is needed. For detailed information on scheduler topologies see the technical paper, ″WebSphere Enterprise Scheduler planning and administration guide″.

Multiple schedulers can share a database when you specify unique table prefix values in each scheduler configuration. This sharing can lower the cost of administering scheduler databases.

Complete the following steps to create scheduler databases.

1. Create a database. To create the database for a scheduler or to determine if an existing database is adequate for a scheduler, review the topic, ″Create scheduler databases″.
2. Create the scheduler tables. There are three methods for creating the tables for a scheduler:

a. Create tables for schedulers using the administrative console. Use the administrative console to add, delete and verify database tables through your Web browser. This method is ideal for developers and simple scheduler topologies.
b. Create tables for schedulers using JMX or scripting.

   Use JMX to add, delete and verify database tables programmatically with Java or scripting. This method is ideal for automating scheduler configurations for simple scheduler topologies.
c. Create tables for schedulers using DDL files. Manually edit the DDL files through your favorite text editor, and verify that mapping between the table names and the scheduler resources and data sources is correct.

Creating scheduler databases:

The scheduler uses the scheduler database for storing and running tasks. To create a scheduler database, your database system must be installed and available.

The performance of schedulers is ultimately limited by the performance of the database. If you need more tasks per second, you can run the scheduler daemons on larger systems or you can use clusters for the session beans used by the tasks. Eventually, however, the task database becomes saturated and you then need a larger or better-tuned database system.

Multiple applications can share a scheduler database. This sharing can lower the cost of administering scheduler databases.

The scheduler requires a database, a JDBC provider, and a data source.

1. Create the database according to the description for your database system:
   - Creating a Cloudscape database for schedulers.
   - Creating a DB2 database for schedulers.
   - Creating a DB2 database for z/OS for schedulers.
   - Creating a DB2 for iSeries database for schedulers.
   - Creating an Informix database for schedulers.
   - Creating a Microsoft SQL Server database for schedulers.
   - Creating an Oracle database for schedulers.
   - Creating a Sybase database for schedulers.
2. If the database is not on the same machine as your IBM WebSphere Application Server, verify that you can access the database from your application server machine.
3. Configure your JDBC provider and data source. For details, see the topic Creating and configuring a JDBC provider and data source. The JDBC driver can be either one-phase or two-phase commit depending on whether other transactions take place using other data sources, for example, while using the scheduler. The data source can represent multiple versions of the product.

The database is created and ready for you to create scheduler tables.

Creating Cloudscape databases for schedulers:

This topic describes how to create Cloudscape databases for schedulers using data definition language (DDL) or structured query language (SQL) files.

To create Cloudscape databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Verify that you have administrator rights for the database system.

3. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in Java code, which results in code page conversion problems when a client uses an incompatible code page.
4. Use the ij utility supplied with the Cloudscape system to create the database. To use ij to create a database called scheddb which is located in /opt, for example, you would do the following:

   ```
   ij.sh
   ij>connect '/opt/scheddb;create=true';
   ij>quit;
   ```

   The embedded version of Cloudscape supports only one local connection. If the Application Server product is running and accessing a Cloudscape database, then any attempts to open a second connection to the database from the command line are rejected.

   Note: Add a semi-colon (;) at the end of each command. Otherwise, ij does not execute the command.
5. Exit the ij utility by issuing the quit command: `quit;`

The Cloudscape database for the scheduler service exists.

Creating DB2 databases for schedulers:

This topic describes how to create DB2 databases for scheduler, using data definition language (DDL) or structured query language (SQL) files.

To create DB2 databases for scheduler, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a DB2 command-line window.
2. Make sure that you have administrator rights for the database system.
3. Verify that the database supports Unicode (UTF-8). Otherwise, it cannot store all the characters that can be handled in Java code, which might result in code page conversion problems, when a client uses an incompatible code page.

   To avoid deadlocks, be sure that the DB2 isolation level is set to ″read stability″. If necessary, enter the command

   ```
   db2set DB2_RR_TO_RS=YES
   ```

   then restart the DB2 instance to activate the change.
4. In the DB2 command line processor, enter this command to create the database with an example name, `scheddb`:

   ```
   db2 CREATE DATABASE scheddb USING CODESET UTF-8 TERRITORY en-us
   ```

   A DB2 database named `scheddb` has been created.

The DB2 database for the scheduler exists.

Creating DB2 for z/OS databases for schedulers:

This topic describes how to create DB2 for z/OS databases for schedulers using data definition language (DDL) or structured query language (SQL) files.

To create DB2 for z/OS databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. You must have a DB2 subsystem installed in the z/OS platform on a UNIX or Windows machine.
2. On the z/OS machine that hosts the database:

a. Log on to the native z/OS environment.
b. If multiple DB2 systems are installed, then decide which subsystem you want to use.
c. Create a storage group and note the name.
d. Decide which user ID is used to connect to the database from the remote machine running the product. Normally, for security reasons, this user ID is not the one you used to create the database.
e. Grant the user ID the rights to access the database and storage group. The user ID must also have permission to create new tables for the database.

3. On the server machine:
   a. Change to the *Scheduler* subdirectory in the application server installation root directory.
   b. Edit the `createTablespaceDB2ZOS.ddl` script. Replace **@STG@** with the storage group name. Replace **@DBNAME@** with the database name (not the subsystem name), and replace **@SCHED_TABLESPACE@** with the name of a valid tablespace.
   c. Run your customized version of `createTablespaceDB2ZOS.ddl`, as described in the header of the script.
   d. To avoid deadlocks, verify that the DB2_RR_TO_RS DB2 flag is set to **YES**. If necessary, restart the DB2 instance to activate the change.

The DB2 for z/OS database for the scheduler service is created.

Creating DB2 for iSeries databases for schedulers:

This topic describes how to create DB2 for iSeries databases for scheduler, using data definition language (DDL) or structured query language (SQL) files.

To create DB2 for iSeries databases for scheduler, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Run the following command to start an interactive SQL session:
   ```
   STRSQL
   ```
2. In interactive SQL, enter this command to create the collection with an example name, `scheddb`:
   ```
   CREATE COLLECTION scheddb
   ```
3. Exit the interactive SQL session.
4. Change the owner for the new collection to QEJBSVR by executing the following command:
   ```
   CHGOBJOWN OBJ(scheddb) OBJTYPE(*LIB) NEWOWN(QEJBSVR)
   ```

   where `scheddb` is the name of the collection you created in the previous step.

The DB2 for iSeries database for the scheduler exists.

Creating Informix databases for schedulers:

This topic describes how to create Informix databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

To create Informix databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Verify that you have administrator rights for the database system.
3. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible code page.

4. If you want to create a new database named `scheddb`, for example, enter the command:
   ```
   dbaccess CREATE DATABASE scheddb with log
   ```

The Informix database for scheduler exists.

Creating Microsoft SQL Server databases for schedulers:

This topic describes how to create Microsoft SQL Server databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

To create Microsoft SQL Server databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Make sure that you are using a user ID that has administrator rights for the database system.
2. In the **Enterprise Manager**, expand a server group, then expand a server. A Microsoft SQL Server database named `scheddb` is created.
3. Right-click **Databases**, then click **New Database**.
4. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible codepage.
5. Type the name `scheddb`.
6. Modify any default values, and save your changes. The Microsoft SQL Server database, scheddb, is created.

The Microsoft SQL Server database for scheduler exists.

Creating Oracle databases for schedulers:

This topic describes how to create Oracle databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

To create Oracle databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible code page.
4. Use the Database Configuration Assistant to create the database, `scheddb`, for example. Verify that you select the **JServer** option for the database. Use a Unicode code page when creating the database. The text data you pass to the APIs must be compatible with the selected code page.

The Oracle database for scheduler exists.

Creating Sybase databases for schedulers:

This topic describes how to create Sybase databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

This topic describes how to create Sybase databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Make sure that you have the DTM option for Sybase ASE installed.

4. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible code page.
5. Use the Sybase isql utility to create the database, `scheddb`, for example. See your Sybase product documentation for details.

The Sybase database for the scheduler exists.

Scheduler table management functions:

The administration console and the WASSchedulerConfiguration MBeans provide simplified methods for creating scheduler tables and schema, verifying that the scheduler tables and schema are setup properly and are accessible and removing scheduler tables and schema.

When connecting to a node agent, the operation attempts to verify, create, or drop the tables based on the most granular scope where the scheduler and associated data source is located.

Note: There are limitations when running the table management functions relating to data source access. See the Verifying a connection topic for details on these limitations. If a connection cannot be verified successfully, the scheduler table management functions will fail.

Verify tables

Validates that scheduler data sources, table prefixes, security authentication information and tables are configured correctly. You can use this verification method in production and development environments without altering database properties.

Create tables

Creates the necessary tables and indices required for schedulers to operate. This method of creating scheduler tables is designed for simple topologies and development environments. Use the supplied scheduler data definition language files for advanced or production environments and for databases that do not support this feature. For details, see the topic Creating scheduler tables using the administrative console.

Drop tables

Specifies the removal of tables and indices required for schedulers to operate. This method of removing scheduler tables and indices is recommended for development environments. When you drop tables, the action removes all previously scheduled tasks, and the scheduler no longer operates successfully, until the tables are recreated.

Scheduler table definition:

Schedulers require database tables and indices with a table prefix. This page provides reference information about the tables.

Each scheduler requires several database tables and indices to operate. Each table name and index described in this topic requires a table prefix. For example, if the scheduler is configured with a table prefix value, **SCHED_**, the table with the name, **TASK**, would be named **SCHED_TASK**. See Scheduler settings for details on the table prefix.

To create the tables, see Creating the database for schedulers. To see the exact schema definition such as field sizes and types, see Creating scheduler tables using DDL files. This section references the location where the DDL or SQL statements are stored. These statements create the table schema.

Note: The information in this topic is provided for problem determination. Do not alter the scheduler table names, field names or index names. The data content format might change without notice. Be aware of this factor when accessing the tables directly. Modifying data in the tables without using the Scheduler API might cause failures.

TASK

The TASK table contains the tasks that have been scheduled, but not yet purged. The primary key for this table is the TASKID which equates to the getTaskID() method on the com.ibm.websphere.scheduler.TaskStatus interface.

Since there is one row in this table for each task, it is important that the database and table support row-locking. Using page, or table locks, inhibits the scheduler from running tasks concurrently.

Field name	Purpose and notes
TASKID	Contains all of the tasks that have been scheduled, but not yet purged. The primary key for this table is TASKID which equates to the getTaskID() method on the com.ibm.websphere.scheduler.TaskStatus interface. Since there is one row in this table for each task, it is important that the database and table support row-locking. Using page, or table locks, will inhibit the scheduler from running tasks concurrently.
VERSION	Internal version ID of this row format.
ROW_VERSION	The version of this row. Used for optimistic locking.
TASKTYPE	The type of task: **1**=BeanTaskInfo, **2**=MessageTaskInfo
TASKSUSPENDED	This value indicates if the task is suspended or if it is running. The task is suspended if the value BITWISE AND 1 equals **1**. The task is running if the value BITWISE AND 2 equals **2**.
CANCELLED	The value, **1**, if the task is cancelled.
NEXTFIRETIME	The date in milliseconds using java.util.Date.getTime() when the task is scheduled to run next.
STARTBYINTERVAL	The start-by-interval of the task.
STARTBYTIME	Reserved.
VALIDFROMTIME	The task start time.
VALIDTOTIME	Reserved.
REPEATINTERVAL	The task repeat interval.
MAXREPEATS	The number of times to run the task.
REPEATSLEFT	The number of times the task has yet to run.
TASKINFO	Internal binary data.
NAME	The task name.
AUTOPURGE	The value, **1**, if the task is to automatically purge upon completion.
FAILUREACTION	Reserved.
MAXATTEMPTS	Reserved.
QOS	Reserved.
PARTITIONID	Reserved.
OWNERTOKEN	The task owner.

CREATETIME	The time in milliseconds using java.util.Date.getTime() when the task was created.

The TASK table also has the following indices that are required to allow the scheduler to run and access tasks concurrently:

- TASK_IDX1 – Used to access individual tasks using the Scheduler API.
- TASK_IDX2 – Used by the poll daemon to load expiring tasks.

TREG

The TREG table is used to store scheduler information that is shared between redundant schedulers. This table is not highly used.

Field name	Purpose and notes
REGKEY	The registry key. This is the primary key of the table.
REGVALUE	The registry value.

LMGR

The LMGR table is used to track the leases that redundant schedulers use. This table is not highly used.

Field name	Purpose and notes
LEASENAME	The name of the lease. This is the scheduler JNDI name and is the primary key.
LEASEOWNER	The owner of the lease. The format is `Cell/Node/Server`.
LEASE_EXPIRE_TIME	The time in milliseconds using java.util.Date.getTime() when the lease for the scheduler expires.
DISABLED	Reserved.

LMPR

The LMPR table is used to store arbitrary properties for the lease. This table is not highly utilized.

Field name	Purpose and notes
LEASENAME	The name of the lease. See the LMGR table.
NAME	The name of the property.
VALUE	The value of the property.

The LMPR table also has the following index:

- LMPR_IDX1 – Used to retrieve properties for a given lease.

Creating scheduler tables using the administrative console:

To create scheduler tables using the administrative console, the scheduler requires a database, a Java DataBase Connectivity (JDBC) provider and a data source.

Note: Limitations for Oracle XA databases

Oracle XA prohibits required schema operations in a global transaction environment. Local transactions are not supported. If you have schedulers that use an Oracle XA data source, either temporarily change the scheduler configuration to use a non-XA Oracle data source, or create the tables manually using the supplied DDL files. If you use the administrative console to create or drop scheduler tables for a scheduler configured to use an Oracle XA data source, then you receive a SchedulerDataStoreException error message, and the operation fails.

Note: Limitations for DB2 z/OS databases

Creating and dropping tables using the administrative console is not supported for DB2 z/OS databases. A database administrator is typically involved with defining and managing databases on DB2 z/OS systems. The administration interface is targeted for the non-database administrator or developer who does not want to know the specifics of setting up the scheduler database. The scheduler has DDL files available for the database administrator to create the required tables.

1. Verify that the database to be used for this scheduler is available and accessible by the application server. Review the Creating scheduler databases and tables topic for instructions on creating a database. The remaining steps describe how to create scheduler tables in an existing database.
2. Start the administrative console.
3. Create a JDBC data source that refers to the scheduler database.
4. Test the data source connection.
5. Create a scheduler. This configuration object contains the desired table prefix and the JNDI name of the JDBC data source. Verify that you save the new Scheduler to the configuration repository before you proceed to the next step.
6. Click **Resources** > **Schedulers** to view all defined schedulers.
7. Select one or more schedulers.
8. Click **Create Tables** to create the tables for the selected schedulers in their associated database. The tables and indices you created reflect the table prefixes and data sources specified in each scheduler configuration.
9. Restart the server or start the poll daemon to run scheduler tasks.

Scheduler tables and schema are created.

Creating scheduler tables using scripting and Java Management Extensions:

Creating scheduler tables using scripting and Java Management Extensions requires a database, a JDBC provider, and a data source.

Note: Limitations for Oracle XA databases

Oracle XA prohibits required schema operations in a global transaction environment. Local transactions are not supported. If you have schedulers that use an Oracle XA data source, either temporarily change the scheduler configuration to use a non-XA Oracle data source, or create the tables manually using the supplied DDL files. If you use the administrative console to create or drop scheduler tables for a scheduler configured to use an Oracle XA data source, then you receive a SchedulerDataStoreException error message, and the operation fails.

Note: Limitations for DB2 z/OS databases

Creating and dropping tables using the administrative console is not supported for DB2 z/OS databases. A database administrator is typically involved with defining and managing databases on DB2 z/OS systems. The administration interface is targeted for the non-database administrator or developer who does not want to know the specifics of setting up the scheduler database. The scheduler has DDL files available for the database administrator to create the required tables.

1. Verify that the database to be used for this Scheduler is available and accessible by the application server. Review the Creating scheduler databases and tables topic for instructions on creating a database. The remainder of these steps describe how to create scheduler tables in an existing database.
2. Launch the wsadmin tool and connect to a deployment manager or application server. This process requires an active server to be available and fails, if you are disconnected from the server.
3. Create a JDBC data source that refers to the scheduler database.
4. Test the data source connection.
5. Create a scheduler. This configuration object contains the desired table prefix and the JNDI name of the JDBC data source. Verify that you save the new Scheduler to the configuration before you proceed to the next step.
6. Run the **createTables** MBean operation.
   a. Look up the SchedulerConfiguration object or use the object you created in a previous step.
   b. Locate the **WASSchedulerConfiguration** MBean.
   c. Run one of the **createTables** MBean operation on the **WASSchedulerConfiguration** object to create the tables for the specified **SchedulerConfiguration** object in its associated database. The tables and indices that you created reflect the table prefix and data source specified in the scheduler configuration.
7. Restart the server or start the poll daemon to run scheduler tasks.

Scheduler tables and schema are created.

Example: Using scripting to verify scheduler tables:

Use the wsadmin scripting tool to invoke a `Jacl` script and verify tables and indices for a scheduler.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which verifies that the tables and indices are created correctly for a scheduler. See the “Configuring Schedulers” topic for details on how a scheduler is created.

```
# Example JACL Script to verify the scheduler tables

# The name of the scheduler to verify
set schedName "My Scheduler"

puts ""
puts "Looking-up Scheduler Configuration Helper MBean"
puts ""
set schedHelper [$AdminControl queryNames WebSphere:*,type=WASSchedulerCfgHelper]

#Access the configuration object.
set myScheduler [$AdminConfig getid /SchedulerConfiguration:$schedName/]

if {$myScheduler == ""} {
    puts ""
    puts "Error: Scheduler $schedName could not be found."
    puts ""
    exit
}

# Invoke the verifyTables method on the helper MBean.

puts ""
puts "Verifying tables for:"
puts "$myScheduler"
puts ""

if { [catch {$AdminControl invoke $schedHelper verifyTables $myScheduler} errorInfo] } {
    puts ""
```

```
    puts "Error verifying tables: $errorInfo"
    puts ""
} else {
    puts ""
    puts "Tables verified successfully."
    puts ""
}
```

Related tasks

“Configuring schedulers” on page 2048
Before your application can make use of the scheduler service, you must configure a scheduler using the administrative console, configuration service or scripting. Conceptually, a scheduler is similar to a data source in that you must specify various configuration attributes, including a JNDI name where the instance is bound. Once defined, an application using the Scheduler API or WASScheduler MBean can look up the scheduler object and call various methods to manage tasks.

“Configuring schedulers using Java Management Extensions” on page 2053
Schedulers can be created or configured using the Java Management Extensions (JMX) API using one of several scripting languages or Java.

“Creating scheduler tables using scripting and Java Management Extensions” on page 2064
Creating scheduler tables using scripting and Java Management Extensions requires a database, a JDBC provider, and a data source.

Related reference

“Scheduler table management functions” on page 2061
The administration console and the WASSchedulerConfiguration MBeans provide simplified methods for creating scheduler tables and schema, verifying that the scheduler tables and schema are setup properly and are accessible and removing scheduler tables and schema.

Example: Using scripting to create scheduler tables:

Use the wsadmin scripting tool to invoke a `Jacl` script and create scheduler tables.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which creates the scheduler tables for a configured scheduler. See the “Configuring Schedulers” topic for details on how to create a scheduler.

```
# Example JACL Script to create the scheduler tables

# The name of the scheduler to create tables for
set schedName "My Scheduler"

puts ""
puts "Looking-up Scheduler Configuration Helper MBean"
puts ""
set schedHelper [$AdminControl queryNames WebSphere:*,type=WASSchedulerCfgHelper]

#Access the configuration object.
set myScheduler [$AdminConfig getid /SchedulerConfiguration:$schedName/]

if {$myScheduler == ""} {
    puts ""
    puts "Error: Scheduler with name: $schedName could not be found."
    puts ""
    exit
}

# Invoke the createTables method on the helper MBean.

puts ""
puts "Creating tables for:"
puts "$myScheduler"
puts ""
```

```
if {[catch {
    set result [$AdminControl invoke $schedHelper createTables $myScheduler]
    if {$result} {
        puts ""
        puts "Successfully created the tables."
        puts ""
    } else {
        puts ""
        puts "The tables were already created."
        puts ""
    }
  } errorInfo ] } {
    puts ""
    puts $errorInfo
    puts ""
}
```

Related tasks

"Configuring schedulers" on page 2048
Before your application can make use of the scheduler service, you must configure a scheduler using the administrative console, configuration service or scripting. Conceptually, a scheduler is similar to a data source in that you must specify various configuration attributes, including a JNDI name where the instance is bound. Once defined, an application using the Scheduler API or WASScheduler MBean can look up the scheduler object and call various methods to manage tasks.

"Configuring schedulers using Java Management Extensions" on page 2053
Schedulers can be created or configured using the Java Management Extensions (JMX) API using one of several scripting languages or Java.

"Creating scheduler tables using scripting and Java Management Extensions" on page 2064
Creating scheduler tables using scripting and Java Management Extensions requires a database, a JDBC provider, and a data source.

Related reference

"Scheduler table management functions" on page 2061
The administration console and the WASSchedulerConfiguration MBeans provide simplified methods for creating scheduler tables and schema, verifying that the scheduler tables and schema are setup properly and are accessible and removing scheduler tables and schema.

Example: Using scripting to drop scheduler tables:

Use the wsadmin scripting tool to invoke a `Jacl` script and remove scheduler tables.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which removes the scheduler tables for a configured scheduler. See the "Configuring Schedulers" topic for details on how a scheduler is created

```
# Example JACL Script to drop the scheduler tables

# The name of the scheduler to drop the tables for
set schedName "My Scheduler"

puts ""
puts "Looking-up Scheduler Configuration Helper MBean"
puts ""
set schedHelper [$AdminControl queryNames WebSphere:*,type=WASSchedulerCfgHelper]

#Access the configuration object.
set myScheduler [$AdminConfig getid /SchedulerConfiguration:$schedName/]

if {$myScheduler == ""} {
    puts ""
    puts "Error: Scheduler with name: $schedName could not be found."
    puts ""
    exit
```

```
}

# Invoke the dropTables method on the helper MBean.

puts ""
puts "Dropping tables for:"
puts "$myScheduler"
puts ""

if {[catch {
    set result [$AdminControl invoke $schedHelper dropTables $myScheduler]
    if {$result} {
        puts ""
        puts "Successfully dropped the tables."
        puts ""
    } else {
        puts ""
        puts "The tables were already dropped."
        puts ""
    }
  } errorInfo ] } {
    puts ""
    puts $errorInfo
    puts ""
}
```

Related tasks

"Configuring schedulers" on page 2048
Before your application can make use of the scheduler service, you must configure a scheduler using the administrative console, configuration service or scripting. Conceptually, a scheduler is similar to a data source in that you must specify various configuration attributes, including a JNDI name where the instance is bound. Once defined, an application using the Scheduler API or WASScheduler MBean can look up the scheduler object and call various methods to manage tasks.

"Configuring schedulers using Java Management Extensions" on page 2053
Schedulers can be created or configured using the Java Management Extensions (JMX) API using one of several scripting languages or Java.

"Creating scheduler tables using scripting and Java Management Extensions" on page 2064
Creating scheduler tables using scripting and Java Management Extensions requires a database, a JDBC provider, and a data source.

Related reference

"Scheduler table management functions" on page 2061
The administration console and the WASSchedulerConfiguration MBeans provide simplified methods for creating scheduler tables and schema, verifying that the scheduler tables and schema are setup properly and are accessible and removing scheduler tables and schema.

Creating scheduler tables using DDL files:

This topic provides the steps for creating scheduler tables using DDL files.

Your database system must be installed and available.

The scheduler requires a database, a JDBC provider, and a data source.

Complete the following steps to create scheduler tables using DDL files.

1. Verify that your database is created. See the topic ″Creating scheduler databases.″
2. Create the database tables according to the instructions for your database system.
   - Creating Cloudscape tables for schedulers.
   - Creating DB2 tables for schedulers.

- Creating DB2 tables for z/OS for schedulers.
- Creating Informix tables for schedulers.
- Creating Microsoft SQL Server tables for schedulers.
- Creating Oracle tables for schedulers.
- Creating Sybase tables for schedulers.

Creating Cloudscape tables for schedulers:

This topic describes how to create tables for schedulers on Cloudscape databases, using data definition language (DDL) or structured query language (SQL) files.

This task requires you to configure a database and make it available. See the ″Creating Cloudscape databases for schedulers″ topic, for more information.

To create tables for schedulers on Cloudscape databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Create the schema.
   a. Using a text editor, edit the script, *%app_server_root%*\Scheduler\createSchemaCloudscape.ddl, according to the instructions at the top of the file.

      Note: When setting the table prefix, capitalize all characters.
   b. Enter one of the following commands.

      Note: Cloudscape provides both an embedded and network server version. This example is for the embedded version of Cloudscape. See the Cloudscape product documentation for more details on running DDL scripts.

      On Windows systems (using the example name, scheddb):

      ```
      %app_server_root%\cloudscape\bin\embedded\ij.bat %app_server_root%\Scheduler\createSchemaCloudscape.ddl
      ```

      On UNIX systems (using the example name, scheddb):

      ```
      %app_server_root%/cloudscape/bin/embedded/ij.sh %app_server_root%/Scheduler/createSchemaCloudscape.ddl
      ```

The Cloudscape tables and schema for the scheduler exist.

Creating DB2 tables for schedulers:

This topic describes how to create tables for scheduler on DB2 databases, using data definition language (DDL) or structured query language (SQL) files.

This task requires you to configure a database and make it available. See the ″Creating DB2 databases for schedulers″ topic for more information.

To create tables for scheduler on DB2 databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a DB2 command-line window.
2. Verify that you have administrator rights for the database system.
3. Create the table space and schema.
   a. Analyze the results of your experiences during development and system testing. The size of your database depends on many factors. If possible, distribute table space containers across different logical disks, and implement an appropriate security policy. Consider the performance implications of your choices for buffer pools and log file settings.
   b. Using a text editor, edit the following scripts according to the instruction at the top of each file.

Note: When setting the table prefix, capitalize all characters.

`%WAS_HOME%\Scheduler\createTablespaceDB2.ddl, %WAS_HOME%\Scheduler\createSchemaDB2.ddl, %WAS_HOME%\Scheduler\dropSchemaDB2.ddl, and %WAS_HOME%\Scheduler\dropTablespaceDB2.ddl.`

c. Verify that you are attached to the correct instance. Check the environment variable DB2INSTANCE.

d. To connect to the database, `scheddb`, for example, and enter the command:

```
db2 connect to scheddb
```

e. Create the table space. Enter the following command:

```
db2 -tf createTablespaceDB2.ddl
```

Verify that the script output contains no errors. If there were any errors, you can drop the table space using the following script:

```
dropTablespaceDB2.ddl
```

f. To create the schema (tables and indices), in the DB2 command line processor, enter the command `db2 -tf createSchemaDB2.ddl`. Verify that the script output contains no errors. If there were any errors, you can use the following file to drop the schema:

```
dropSchemaDB2.ddl
```

g. Verify that the DB2_RR_TO_RS DB2 flag is set to **YES** to avoid deadlocks. Restart the DB2 instance to activate the change, if needed.

The DB2 tables and schema for the scheduler exist.

Creating DB2 for z/OS tables for schedulers:

This topic describes how to create tables for a scheduler on a DB2 for z/OS database using data definition language (DDL) or structured query language (SQL) files.

This task requires that you configure a database and make it available. See the ″Creating DB2 for z/OS databases for schedulers″ topic for more information.

In addition, you must have the following two machines:

1. The z/OS machine that is hosting the database
2. The WebSphere Application Server machine that is running the scheduler

To create tables for a scheduler on a DB2 for z/OS database, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Work with the z/OS machine that hosts the database to:
   a. Log on to the native z/OS environment.
   b. Decide which subsystem you want to use, if multiple DB2 systems are installed.
   c. Note of the Internet Protocol (IP) port to which the DB2 subsystem is listening.
   d. Use the DB2 administration menu to create a new database named, for example, SCHEDDB. Note the database name.
   e. Create a storage group and note the name.
   f. Decide which user ID is used to connect to the database from the remote machine running the product. Normally, for security reasons, this user ID is not the one you used to create the database.
   g. Grant the user ID the rights to access the database and storage group. The user ID must also have permission to create new tables for the database.
2. Work with the Application Server machine to:
   a. Verify that you have DB2 Connect Gateway (Version 8.1 fix pack 3 or higher) installed. This component is part of the DB2 UDB ESE package; however, you can also install it separately.

b. Catalog the remote database using the following commands, either in a script or in a DB2 command line window:

```
catalog tcpip node zosnode remote hostname server IP_port ostype mvs;
catalog database subsystem as subsystem at node zosnode authentication dcs;
catalog dcs database subsystem as subsystem parms ',,INTERRUPT_ENABLED'
```

An important difference exists between DB2 UDB and DB2 for z/OS. DB2 UDB does not have the concept of a subsystem, but DB2 for z/OS does have subsystems. To avoid confusion between Database name and Subsystem name, remember that because DB2 for z/OS runs in a subsystem, the `catalog node` and `catalog database` commands must identify the appropriate subsystem. On DB2 UDB, the subsystem name is not a known concept, and the database name to which it connects is actually the name of the DB2 for z/OS subsystem.

c. Verify that you can establish a connection to the remote subsystem by entering the following command:

```
db2 connect to subsystem user userid using password
```

d. Change to the scheduler subdirectory in the application server installation root directory.

e. Edit the `createTablespaceDB2ZOS.ddl` script. Replace `@STG@` with the storage group name. Replace `@DBNAME@` with the database name (not the subsystem name), and replace `@SCHED_TABLESPACE@` with the name of a valid table space. After you replace the database name, place it into an existing JCL and run the job.

f. Run your customized version of `createTablespaceDB2ZOS.ddl`, as described in the header of the script. If this script does not work, or if you want to remove the table space, edit and run the `dropTablespaceDB2ZOS.ddl` script.

g. Edit the `createSchemaDB2ZOS.ddl` script. Replace `@STG@` with the storage group name. Replace `@DBNAME@` with the database name (not the subsystem name). Replace `@TABLE_PREFIX@` with the Table Prefix in the configured scheduler resource, and replace `@SCHED_TABLESPACE@` with a valid table space that was created by the `createTablespaceDB2ZOS.ddl` script.

Note: When setting the table prefix, capitalize all characters.

h. Run your customized version of the `createSchemaDB2ZOS.ddl` script, as described in the header of the script. If this script does not work, or if you want to remove the tables and views, use `dropSchemaDB2ZOS.ddl` to drop the schema.

i. To avoid deadlocks, verify that the DB2_RR_TO_RS DB2 flag is set to **YES**. If necessary, restart the DB2 instance to activate the change. In addition, verify that the table space was created with the LOCKSIZE ROW statement.

The DB2 for z/OS tables and schema for the scheduler exist.

Creating Informix tables for schedulers:

This topic describes how to create tables for schedulers on Informix databases, using data definition language (DDL) or structured query language (SQL) files.

This task requires that you configure a database and make it available. See the ″Creating Informix databases for schedulers″ topic for more information.

To create tables for schedulers on Informix databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Verify that you have administrator rights for the database system.
3. Create the schema.
   a. Using a text editor, edit the script `%WAS_HOME%\Scheduler\createSchemaInformix.sql` according to the instruction at the top of the file.

Note: When setting the table prefix, capitalize all characters.

b. Enter the following command, using the database, `scheddb`, for example:

```
dbaccess scheddb createSchemaInformix.sql
```

The Informix tables and schema for the scheduler exist.

Creating Microsoft SQL Server tables for schedulers:

This topic describes how to create tables for schedulers on Microsoft SQL Server databases, using data definition language (DDL) or structured query language (SQL) files.

This task requires you to configure a database and make it available. See the ″Creating Microsoft SQL Server databases for schedulers″ topic for more information.

To create tables for schedulers on Microsoft SQL Server databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Change to the directory where the configuration scripts for scheduler are located. This is the Scheduler subdirectory of the IBM WebSphere Application Server installation directory.

   On Windows systems, enter:

   ```
   cd %app_server_root%\Scheduler
   ```

   On UNIX systems, enter:

   ```
   cd $app_server_root/Scheduler
   ```
3. Use a text editor to edit the schema creation script, `createSchemaMSSQL.sql`, according to the instructions at the beginning of the file.

   Note: When setting the table prefix, capitalize all characters.
4. Create the schema:
   a. Verify that you have administrator rights for the database system. The user ID you use to create the schema must be the one that you configure WebSphere Application Server to use when accessing the database.
   b. Run the following script to create the schema (tables and views) :

   ```
   isql -S <servername> -U<userid> -P<password> -D<databaseName> -i<script name>
   ```

The Microsoft SQL Server tables and schema for scheduler exist.

Creating Oracle tables for schedulers:

This topic describes how to create tables for schedulers on Oracle databases, using data definition language (DDL) or structured query language (SQL) files.

This task requires you to configure a database and make it available. See the ″Creating Oracle databases for schedulers″ topic for more information.

To create tables for schedulers on Oracle databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Create the table space and schema.
   a. Using a text editor, edit the following scripts according to the instructions at the top of the files.

      Note: When setting the table prefix, capitalize all characters.

%app_server_root%\Scheduler\createTablespaceOracle.ddl and *%app_server_root%*\Scheduler\createSchemaOracle.ddl

b. Set the environment variable ORACLE_SID, if you do not want the schema to be created in the default instance.

c. Run the script, createTablespaceOracle.ddl, to create the table space.

   For test purposes, use the same location for all table spaces and pass the path as a command line argument to the script. For example, on Windows systems, the user ID is scheduser, password is schedpwd, database name is scheddb, and table space path is d:\mydb\ts. Enter the command: sqlplus scheduser/schedpwd@scheddb @createTablespaceOracle.ddl d:\mydb\ts If you get any errors creating the table space, you can use dropTablespaceOracle.ddl to drop the table space.

d. Run the script, createSchemaOracle.ddl, to create the schema. For example, on Windows systems, enter the following script:

   ```
   sqlplus scheduser/schedpwd@scheddb @createSchemaOracle.ddl
   ```

   If you see any errors creating the schema (tables and views), you can drop the schema by running script:

   ```
   dropSchemaOracle.ddl
   ```

The Oracle tables and schema for scheduler exist.

Creating Sybase tables for schedulers:

This topic describes how to create tables for schedulers on Sybase databases, using data definition language (DDL) or structured query language (SQL) files.

This task requires you to configure a database and make it available. See the ″Creating Sybase databases for schedulers″ topic for more information.

To create tables for schedulers on Sybase databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Make sure that you have the Distributed Transaction Management (DTM) option for Sybase ASE installed.
   a. Set enable DTM to **1** in the Sybase server configuration.
   b. Set enable xact coordination to **1** in the Sybase server configuration.
   c. Add the **dtm_tm_role** role to the Sybase administration user ID. For example, enter the user ID sa.
   d. Restart the Sybase server.
4. Use the Sybase isql utility to create a database. For example, enter the database name scheddb. See your Sybase product documentation for details.
5. Create the schema:
   a. Using a text editor, edit the following script according to the instructions located at the top of the file.

      Note: When setting the table prefix, capitalize all characters.

      ```
      <app_server_root>\Scheduler\createSchemaSybase12.ddl
      ```

   b. Enter the following command:

      ```
      isql -S <servername> -U <userid> -P <password> -D scheddb -i createSchemaSybase12.ddl
      ```

The Sybase tables and schema for scheduler exist.

Startup beans

Using startup beans

There are two types of startup beans: application startup beans and Module startup beans.

A module startup bean is a session bean that is loaded when an EJB Jar file starts. Module startup beans enable Java 2 Platform Enterprise Edition (J2EE) applications to run business logic automatically, whenever an EJB module starts or stops normally. An application startup bean is a session bean that is loaded when an application starts. Application startup beans enable Java 2 Platform Enterprise Edition (J2EE) applications to run business logic automatically, whenever an application starts or stops normally.

Startup beans are especially useful when used with asynchronous bean features. For example, a startup bean might create an alarm object that uses the Java Message Service (JMS) to periodically publish heartbeat messages on a well-known topic. This enables clients or other server applications to determine whether the application is available.

1. For Application startup beans, use the home interface, com.ibm.websphere.startupservice.AppStartUpHome, to designate a bean as an Application startup bean. For Module startup beans, use the home interface, com.ibm.websphere.startupservice.ModStartUpHome, to designate a bean as a Module startup bean.
2. For Application startup beans, use the remote interface, com.ibm.websphere.startupservice.AppStartUp, to define start() and stop() methods on the bean. For Module startup beans, use the remote interface, com.ibm.websphere.startupservice.ModStartUp, to define start() and stop() methods on the bean.

   The startup bean start() method is called when the module or application starts and contains business logic to be run at module or application start time.

   The start() method returns a boolean value. **True** indicates that the business logic within the start() method ran successfully. Conversely, **False** indicates that the business logic within the start() method failed to run completely. A return value of False also indicates to the Application server that application startup is aborted.

   The startup bean stop() methods are called when the module or application stops and contains business logic to be run at module or application stop time. Any exception thrown by a stop() method is logged only. No other action is taken.

   The start() and stop() methods must never use the TX_MANDATORY transaction attribute. A global transaction does not exist on the thread when the start() or stop() methods are invoked. Any other TX_* attribute can be used. If TX_MANDATORY is used, an exception is logged, and the application start is aborted.

   The start() and stop() methods on the remote interface use **Run-As** mode. **Run-As** mode specifies the credential information to be used by the security service to determine the permissions that a principal has on various resources. If security is on, the **Run-As** mode needs to be defined on all of the methods called. The identity of the bean without this setting is undefined.

   There are no restrictions on what code the start() and stop() methods can run, since the full Application Server programming model is available to these methods.
3. Use an *optional* environment property integer, `wasStartupPriority`, to specify the start order of multiple startup beans in the same Java Archive (JAR) file. If the environment property is found and is the wrong type, application startup is aborted. If no priority value is specified, a default priority of 0 is used. It is recommended that you specify the priority property. Beans that have specified a priority are sorted using this property. Beans with numerically lower priorities are run first. Beans that have the same priority are run in an undefined order. All priorities must be positive integers. Beans are stopped in the opposite order to their start priority. The priority values for module startup beans and application startup beans are mutually exclusive. All modules will be started prior to the application being declared as ″started″ and therefore the start() methods for module startup beans within an application will be

invoked prior to the start() methods for any application startup beans. Likewise, all application startup bean stop() methods for a specific Java Archive (JAR) file will be invoked prior to any module startup bean stop() methods for that JAR.

4. For module startup beans, the order in which EJB modules are started can be adjusted via the ″Starting weight″ value associated with each module
5. To control who can invoke startup bean methods via WebSphere Security do the following:
   a. Define the method permissions for the Start() and Stop() methods as you would for any EJB. (See ″Defining method permissions for EJB modules.″)
   b. Ensure that the User that is mapped to the Security Role defined for the startup bean methods is the same user that is defined as the ″Server user ID″ within the User Registry.

View the startup beans service settings.

Startup beans service settings

Use this page to enable startup beans that control whether application-defined startup beans function on this server. Startup beans are session beans that run business logic through the invocation of start and stop methods when applications start and stop. If the startup beans service is disabled, then the automatic invocation of the start and stop methods does not occur for deployed startup beans when the parent application starts or stops. This service is disabled by default. Enable this service only when you want to use startup beans. Startup beans are especially useful when used with asynchronous beans.

To view this administrative console page, click **Servers** > **Application servers** >*server_name* > **Container services** > **Startup beans service**.

Enable service at server startup:

Specifies whether the server attempts to initiate the startup beans service.

Default
Cleared

Range

Selected
When the application server starts, it attempts to initiate the startup bean service automatically.

Cleared
The server does not try to initiate the startup beans service. All startup beans do not start or stop with the application. If you use startup beans on this server, then the system administrator must start the startup beans service manually or select this property, and then restart the server.

Enabling startup beans in the administrative console

Enabling startup beans in the administrative console enables Java 2 Platform Enterprise Edition (J2EE) applications to run business logic automatically, whenever an application starts or stops normally.

Use the following steps to enable startup beans in the administrative console.

1. Start the administrative console.
2. Select **Servers** > **Application Servers** > *server_name* > **Container Services** > **Startup beans service**.
3. Select the **Enable service at server startup** check box.
4. Click **Apply** to save the configuration.

View the startup beans service settings.

Work area

Task overview: Implementing shared work areas

The work area service enables application developers to implicitly propagate information beyond the information passed in remote calls. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of each method. The methods on the server side can use or ignore the information in the work area as appropriate.

Before proceeding with the steps to implement work areas, as described below, review the topic Work area service: Overview.

1. Developing applications that use work areas. Applications interact with the work area service by implementing the UserWorkArea interface.
2. Managing work areas. The work area service is managed using the administrative console.

Overview of work area service

One of the foundations of distributed computing is the ability to pass information, typically in the form of arguments to remote methods, from one process to another. When application-level software is written over middleware services, many of the services rely on information beyond that passed in the application's remote calls. Such services often make use of the implicit propagation of private information in addition to the arguments passed in remote requests; two typical users of such a feature are security and transaction services. Security certificates or transaction contexts are passed without the knowledge or intervention of the user or application developer. The implicit propagation of such information means that application developers do not have to manually pass the information in method invocations, which makes development less error-prone, and the services requiring the information do not have to expose it to application developers. Information such as security credentials can remain secret.

The work area service gives application developers a similar facility. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of every method. The methods on the server side can use or ignore the information in the work area as appropriate. If methods in a server receive a work area from a client and subsequently invoke other remote methods, the work area is transparently propagated with the remote requests. When the creating application is done with the work area, it terminates it.

There are two prime considerations in deciding whether to pass information explicitly as an argument or implicitly by using a work area. These considerations are:

- Pervasiveness: Is the information used in a majority of the methods in an application?
- Size: Is it reasonable to send the information even when it is not used?

When information is sufficiently pervasive that it is easiest and most efficient to make it available everywhere, application programmers can use the work area service to simplify programming and maintenance of code. The argument does not need to go onto every argument list. It is much easier to put the value into a work area and propagate it automatically. This is especially true for methods that simply pass the value on but do nothing with it. Methods that make no use of the propagated information simply ignore it.

Work areas can hold any kind of information, and they can hold an arbitrary number of individual pieces of data, each stored as a property.

Work area property modes: The information in a work area consists of a set of properties; a property consists of a key-value-mode triple. The key-value pair represents the information contained in the property; the key is a name by which the associated value is retrieved. The mode determines whether the property can be removed or modified.

Property modes

There are four possible mode values for properties, as shown in the following code example:

Code example: The PropertyModeType definition

```
public final class PropertyModeType {
   public static final PropertyModeType normal;
   public static final PropertyModeType read_only;
   public static final PropertyModeType fixed_normal;
   public static final PropertyModeType fixed_readonly;
};
```

A property's mode determines three things:
- Whether the value associated with the key can be modified
- Whether the property can be deleted
- Whether the mode associated with the key-value pair can be modified

The two read-only modes forbid changes to the information in the property; the two fixed modes forbid deletion of the property.

The work area service does not provide methods specifically for the purpose of modifying the value of a key or the mode associated with a property. To change information in a property, applications simply rewrite the information in the property; this has the same effect as updating the information in the property. The mode of a property governs the changes that can be made. Modifying key-value pairs describes the restrictions each mode places on modifying the value and deleting the property. Changing modes describes the restrictions on changing the mode.

Changing modes

The mode associated with a property can be changed only according to the restrictions of the original mode. The read-only and fixed read-only properties do not permit modification of the value or the mode. The fixed normal and fixed read-only modes do not allow the property to be deleted. This set of restrictions leads to the following permissible ways to change the mode of a property within the lifetime of a work area:
- If the current mode is normal, it can be changed to any of the other three modes: fixed normal, read-only, fixed read-only.
- If the current mode is fixed normal, it can be changed only to fixed read-only.
- If the current mode is read-only, it can be changed only by deleting the property and re-creating it with the desired mode.
- If the current mode is fixed read-only, it cannot be changed.
- If the current mode is not normal, it cannot be changed to normal. If a property is set as fixed normal and then reset as normal, the value is updated but the mode remains fixed normal. If a property is set as fixed normal and then reset as either read-only or fixed read-only, the value is updated and the mode is changed to fixed read-only.

Note: The key, value, and mode of any property can be effectively changed by terminating (completing) the work area in which the property was created and creating a new work area. Applications can then insert new properties into the work area. This is not precisely the same as changing the value in the original work area, but some applications can use it as an equivalent mechanism.

Nested work areas: Applications can nest work areas. When an application creates a work area, a work area context is associated with the creating thread. If the application thread creates another work area, the new work area is nested within the existing work area and becomes the current work area. Nested work areas allow applications to define and scope properties for specific tasks without having to make them available to all parts of the application. All properties defined in the original, enclosing work area are visible to the nested work area. The application can set additional properties within the nested work area that are not part of the enclosing work area.

An application working with a nested work area does not actually see the nesting of enclosing work areas. The current work area appears as a flat set of properties that includes those from enclosing work areas. In the figure below, the enclosing work area holds several properties and the nested work area holds additional properties. From the outermost work area, the properties set in the nested work area are not visible. From the nested work area, the properties in both work areas are visible.

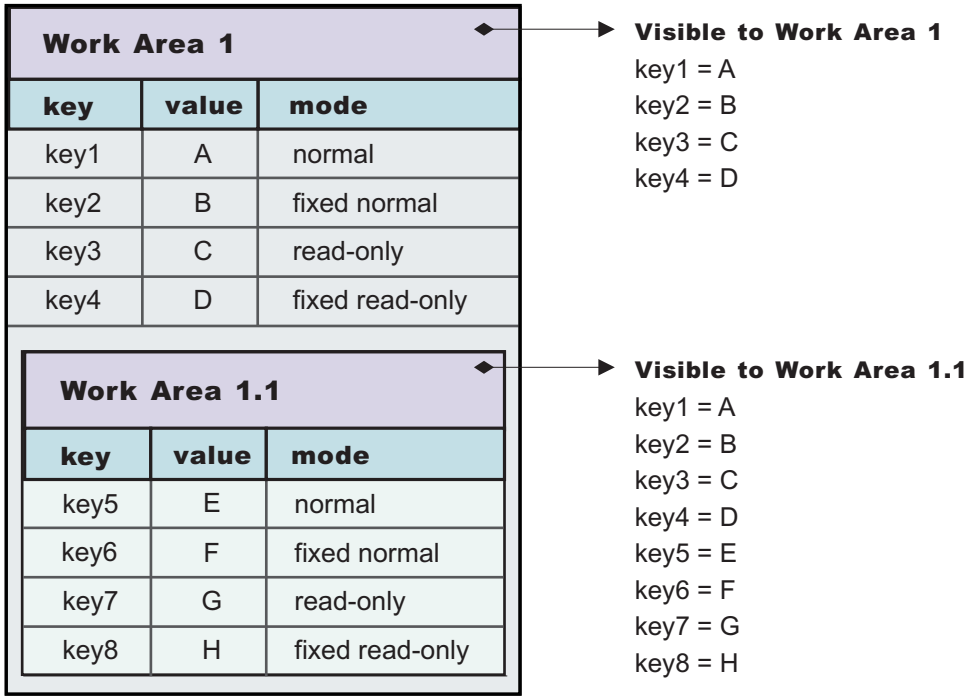


Figure 21. Defining new properties in nested work areas

Nesting can also affect the apparent settings of the properties. Properties can be deleted from or directly modified only within the work areas in which they were set, but nested work areas can also be used to temporarily override information in the property without having to modify the property. Depending on the modes associated with the properties in the enclosing work area, the modes and the values of keys in the enclosing work area can be overridden within the nested work area.

The mode associated with a property when it is created determines whether nested work areas can override the property. From the perspective of a nested work area, the property modes used in enclosing work areas can be grouped as follows:

- Modes that permit a nested work area to override the mode or the value of a key locally. The modes that permit overriding are:
  - Normal
  - Fixed normal
- Modes that do not permit a nested work area to override the mode or the value of a key locally. The modes that do not permit overriding are:
  - Read-only
  - Fixed read-only

If an enclosing work area defines a property with one of the modes that can be overridden, a nested work area can specify a new value for the key or a new mode for the property. The new value or mode becomes the value or mode seen by subsequently nested work areas. Changes to the mode are governed by the restrictions described in Changing modes. If an enclosing work area defines a property with one of the modes that cannot be overridden, no nested work area can specify a new value for the key.

A nested work area can delete properties from enclosing work areas, but the changes persist only for the duration of the nested work area. When the nested work area is completed, any properties that were added in the nested area vanish and any properties that were deleted from the nested area are restored.

The following figure illustrates the overriding of properties from an enclosing work area. The nested work area redefines two of the properties set in the enclosing work area. The other two cannot be overridden. The nested work area also defines two new properties. From the outermost work area, the properties set or redefined in the nested work are not visible. From the nested work area, the properties in both work areas are visible, but the values seen for the redefined properties are those set in the nested work area.

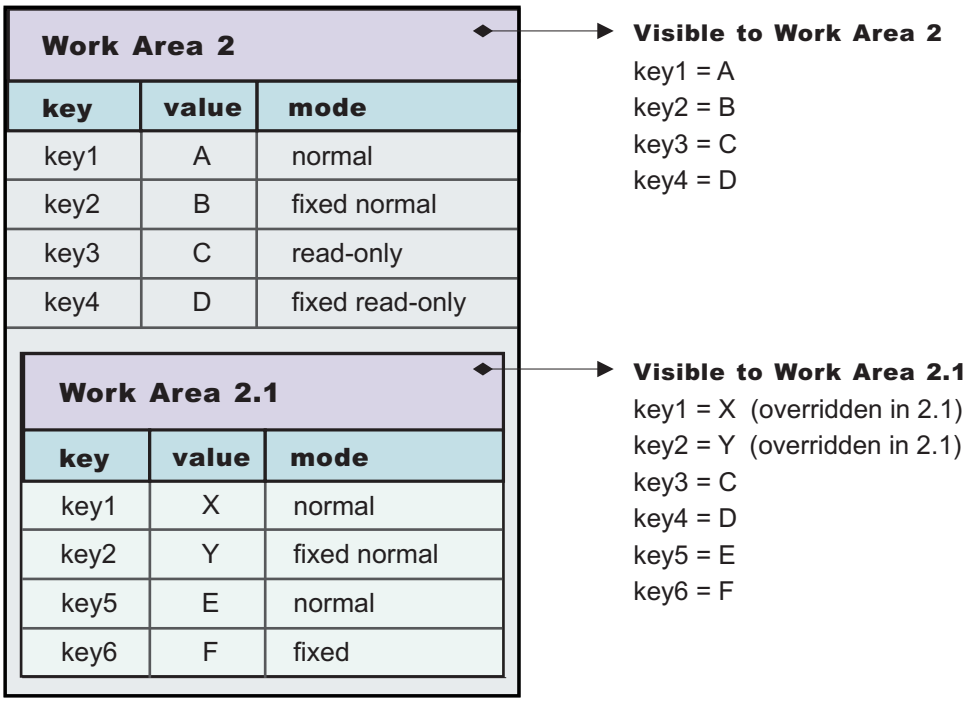


Figure 22. Redefining existing properties in nested work areas

Distributed work areas: The propagation of work area context operates differently depending on whether a work area partition is defined as bidirectional or not. In either case all work area context propagates to a target object on a remote invocation. However, whether the context propagates from a target object back to the originator depends on whether a partition is defined as bidirectional.

Non-bidirectional work area partitions (UserWorkArea partition)

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information is propagated to any remote objects it invokes. However, no changes made to a nested work area on a target object are propagated back to the calling object. The caller's work area is unaffected by changes made in the remote method.

Bidirectional work area partitions

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work

area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, this information is propagated to any remote objects it invokes. In a partition that is not defined as bidirectional, a target application must begin a nested work area before making changes to the imported work area. However, if a partition is defined as bidirectional, a target application need not begin a nested work area before operating on an imported work area. By not beginning a nested work area, any new context set into the work area, or any context changes made by the target application, is not only propagated on future remote invocations but is also propagated back to the originating application (that is, the one who initiated the remote invocation) thus allowing bidirectional propagation of work area context. If the target application does not want new or changed context to propagate back to the originating application, then the target application must begin a nested work area to scope the context to its process. However, the new or changed context in the nested work area propagates on any future remote invocation the target application may make.

WorkArea service: Special considerations: Developers who use work areas should consider the following issues that could potentially cause problems: interoperability between the EJB and CORBA programming models; and the use of work areas with Java's Abstract Windowing Toolkit.

EJB and CORBA interoperability

Although the work area service can be used across the EJB and CORBA programming models, many composed data types cannot be successfully used across those boundaries. For example, if a SimpleSampleCompany instance is passed from the WebSphere environment into a CORBA environment, the CORBA application can retrieve the SimpleSampleCompany object encapsulated within a CORBA Any object from the work area, but it cannot extract the value from it. Likewise, an IDL-defined struct defined within a CORBA application and set into a work area is not readable by an application using the UserWorkArea class.

best-practices: Applications can avoid this incompatibility by directly setting only primitive types, like integers and strings, as values in work areas, or by implementing complex values with structures designed to be compatible, like CORBA valuetypes.

Also, CORBA Anys that contains either the tk_null or tk_void typecode can be set into the work area by using the CORBA interface. However, the work area specification cannot allow the Java 2 Platform, Enterprise Edition (J2EE) implementation to return null on a lookup that retrieves these CORBA-set properties without incorrectly implying that there is no value set for the corresponding key. For example, when a user attempts to retrieve a nonexistent key from a work area, the work area service returns null to indicate that the specified key does not contain a value, implying that the key itself is not in use or does not exist. In the case where CORBA Anys contains either tk_null or tk_void, when a user requests the key associated with one of these values, the work area service returns null as expected. In this case, the key may actually exist and the work area service was simply returning the key's value of null. Therefore, when working with CORBA Anys, a user must not make any implications when a null is returned from a work area because it could mean that either there isn't a property associated with the given key, or that there is a property associated with the given key and it contains a tk_null or tk_void, for example, a null in the J2EE environment. If a J2EE application tries to retrieve CORBA-set properties that are non-serializable, or contain CORBA nulls or void references, the com.ibm.websphere.workarea.IncompatibleValue exception is raised.

Using work areas with Java's Abstract Windowing Toolkit (AWT)

Work areas must be used cautiously in applications that use Java's Abstract Windowing Toolkit (AWT). The AWT implementation is multithreaded, and work areas begun on one thread are not available on another. For example, if a program begins a work area in response to an AWT event, such as pressing a button, the work area might not be available to any other part of the application after the execution of the event completes.

Work area service performance considerations: The work area service is designed to address complex data passing patterns that can quickly grow beyond convenient maintenance. A *work area* is a note pad that is accessible to any client that is capable of looking up Java Naming Directory Interface (JNDI). After a work area is established, data can be placed there for future use in any subsequent method calls to both remote and local resources.

You can utilize a work area when a large number of methods require common information or if information is only needed by a method that is significantly further down the call graph. The former avoids the need for complex parameter passing models where the number of arguments passed becomes excessive and hard to maintain. You can improve application function by placing the information in a work area and subsequently accessing it independently in each method, eliminating the need to pass these parameters from method to method. The latter case also avoids unnecessary parameter passing and helps to improve performance by reducing the cost of marshalling and de-marshalling these parameters over the Object Request Broker (ORB) when they are only needed occasionally throughout the call graph.

When attempting to maximize performance by using a work area, cache the UserWorkArea partition that is retrieved from JNDI wherever it is accessed. You can reduce the time spent looking up information in JNDI by retrieving it once and keeping a reference for the future. JNDI lookup takes time and can be costly.

Additional caching mechanisms available to a user-defined partition are defined by the configuration property, ″Deferred Attribute Serialization″. This mechanism attempts to minimize the number of serialization and deserialization calls. See “Work area partition service” on page 2085 for further explanation of this configuration attribute.

The maxSendSize and maxReceiveSize configuration parameters can affect the performance of the work area. Setting these two values to 0 (zero) effectively turns off the policing of the size of context that can be sent in a work area. This action can enhance performance, depending on the number of nested work areas an application uses. In applications that use only one work area, the performance enhancement might be negligible. In applications that have a large number of nested work areas, there might be a performance enhancement. However, a user must note that by turning off this policing it is possible that an extremely large amount of data might be sent to a server.

Performance is degraded if you use a work area as a direct replacement to passing a single parameter over a single method call. The reason is that you incur more overhead than just passing that parameter between method calls. Although the degradation is usually within acceptable tolerances and scales similarly to passing parameters with regard to object size, consider degradation a potential problem before utilizing the service. As with most functional services, intelligent use of the work areas yields the best results.

The work area service is a tool to simplify the job of passing information from resource to resource, and in some cases can improve performance by reducing the overhead that is associated with a parameter passing when the information is only sparsely accessed within the call graph. Caching the instance retrieved from JNDI is important to effectively maximize performance during runtime.

Managing the work area service - the UserWorkArea partition

The work area service is managed using the administrative console. There are two administrative tasks associated with work areas:

- Enabling the work area service. The work area service is disabled by default on servers but enabled by default on the client
- Managing the size of work areas. Applications can set maximum sizes on each work area to be sent and to be accepted.

Enabling the work area service - the UserWorkArea partition

For an application to take advantage of work areas, the work area service must be enabled for both clients and servers. On a server the service is disabled by default. On the client, the service is enabled by default.

For an application to take advantage of the default partition, UserWorkArea partition, this partition must be enabled by enabling the work area service for both clients and servers. The work area service on a server is disabled by default and the work area service on a client is enabled by default. Note that rather than using this default work area partition, a user can create their own work area partition using the “Work area partition service” on page 2085. For a description of the differences between these two services please see Understand the difference between the work area service and the work area partition service.

1. Enable (or disable) the use of the UserWorkArea partition on a server:
   a. Start the administrative console.
   b. Select **Servers > Application servers >***server_name* **> Business Process Services > Work area service**.
   c. Select or clear the **Startup** check box. This specifies whether or not the server should automatically start the work area service when the server starts.
   d. Save the new configuration and restart the server to apply the new configuration.
2. Enable (or disable) the UserWorkArea partition on a client: Set the com.ibm.websphere.workarea.enabled property to TRUE or FALSE before starting the client. For example, to disable the work area service, when invoking the launchClient script found in the *install_root*/bin directory, add the following system property to the launchClient invocation:

   ```
   -CCDcom.ibm.websphere.workarea.enabled=false
   ```

   Alternatively, this property can be set in a property file that is used by the launchClient script. See “Running application clients” on page 242 for additional information.

After enabling the UserWorkArea partition, manage the size of the work areas that this server can send and the number of work areas that this server can accept. See the article, “Managing the size of work areas” on page 2083.

Work area service settings:

Use this page to manage the work area service.

The work area service manages the scope and implicit propagation of application context. The work area service panel in the administrative console configures the UserWorkArea partition only and has no effect on the work area partition service panel.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Business Process Services > Work area service** .

For additional information about work area, see the com.ibm.websphere.workarea package in the API documentation. The generated API documentation is available in the information center table of contents from the path **Reference** > **Developer** > **API documentation** > **Application programming interfaces**.

Enable service at server startup:

Specifies whether the server attempts to start the work area service.

Selected
When the application server starts, it attempts to start the work area service automatically.

Cleared
The server does not try to start the work area service. If work areas are used on this application server, the system administrator must start the service manually or select this property and then restart the server.

Maximum send size:

Specifies the maximum size of data that can be sent within a single work area.

Data type	Integer
Units	Bytes
Default	10000
Range	-1, 0 (no limit) and 1 to 2147483647

The following values are also used to define the maximum send size.

-1	Default.
0	No limit.

Maximum receive size:

Specifies the maximum size of data that a single work area can receive.

Data type	Integer
Units	Bytes
Default	10000
Range	-1, 0 (no limit) and 1 to 2147483647

The following values are also used to define the maximum receive size.

-1	Default.
0	No limit.

Enable Web service propagation:

Specifies whether the work area is propagated on Web service requests. This option is disabled by default.

Managing the size of work areas

Applications can set maximum sizes on each work area that is sent or received. By default, the maximum size of a work area that is sent by a client and received, then possibly resent, by a server is 32,768 bytes. The maximum size that you can specify is determined by the maximum value expressible in the Java Integer data type, 2,147,483,647. The smallest maximum size that you can specify is 1. Using a maximum size of 1 byte effectively means that no requests associated with the work area can leave the system or enter another system. A value of 0 means that no limit is imposed. A value of -1 means that the default value is to be honored. The default value is also used if an invalid value or a malformed property is specified. You can change this size as described in this topic.

1. Change the size of the work area that can be sent or received by a server:
   a. Start the administrative console.
   b. Select **Servers > Application servers >***server_name* **> Business Process Services** .

- To change the send size or receive size on the work area service (namely the ″UserWorkArea″ partition):
  - Select **Work area service**.
- To change the send size or receive size on a user defined partition:
  - Select **Work area partition service**.
  - Select a partition.

c. Enter a new value in the **Maximum send size** field to modify the size of the work area that this server can send, or enter a new value in the **Maximum receive size** field to modify the size of the work area that this server can accept.

d. Save the new configuration and restart the server to apply the new configuration.

2. Change the size of the work area that can be sent by a client. This step only applies to the UserWorkArea partition on the client. To set the maximum send or receive size on a user defined partition, you must set these values when creating the partition on the client. For more information on creating a partition on a client, see the client section in “Configuring work area partitions.” To change the size of the work area that can be sent by a client, set the com.ibm.websphere.workarea.maxSendSize property to the desired number of bytes before starting the client. You can set the maximum send size as follows:
   - Set the maximum send size when invoking the launchClient invocation script found in the `$WAS_HOME/bin` directory. For example, to set the maximum size to 10,000 bytes, add the following system properties to the launchClient invocation as needed: `-CCDcom.ibm.websphere.workarea.maxSendSize=10000`
   - Set the maximum send size propery, com.ibm.websphere.workarea.maxSendSize, in a property file that is used by the launchClient script. See ″“Running application clients” on page 242″ for additional information.

   Since the UserWorkArea partition is defined as unidirectional, for example, context only propagates on outbound calls and not on the return of those calls, setting the maximum receive size is ignored.

Configuring work area partitions

The work area partition service extends the work area service by allowing the creation of multiple work areas with more configuration options than what is available to the UserWorkArea partition. Follow these steps to create and configure a work area partition:

1. Create a user defined partition on the server.
   a. Start the administrative console.
   b. Click **Servers > Application servers >** *server_name* **> Business process services > Work area partition service**.
   c. Click **New**.
   d. On the settings page for work area partitions, specify values such as the partition name, maxSendSize and maxReceiveSize, then click OK.
   e. Save the new configuration.
   f. Restart the server to apply the new configuration.
2. Create a user defined partition on the client. Use the createWorkAreaPartition method described in the “The Work area partition manager interface” on page 2087 article to programmatically create a partition. See “Example: Work area partition manager” on page 2089 for an example of using this method.

You have created a work area partition.

Retrieve the partition through the work area partition manager interface and use it as defined by the work area service and the UserWorkArea interface. See the topic “Example: Work area partition manager” on page 2089 for an example.

Work area partition service

The work area partition service is an extension of the work area service that allows the creation of multiple custom work areas. The work area partition service is an optional service to users. Any user that currently uses the work area service and the UserWorkArea partition can continue using it in the same manner. The UserWorkArea partition is created automatically (if it has not been disabled) by the work area partition service. By allowing a user the option to create their own work area partition through the work area partition service, they can have more control over configuration and access to their partition.

Unlike the UserWorkArea partition, which is publicly known, work areas created by the work area partition service are accessible to, and known only by the creator. However, the work area partition service does not strictly enforce that a partition is accessed and/or operated on exclusively by the partition creator. There are no limitations should the creator want to publish their work area partition and make it publicly available by binding their partition reference in java naming or by other means. However, the work area partition service does try to hide a partition as much as possible should a user not want others to know about a certain partition. The work area partition service does not allow a person to determine, or query the names of all the partitions that have been created; however, it does not restrict the partitions from being accessed by users other than the creator of that partition. The context of a partition, such as the UserWorkArea partition or a user defined partition, is scoped to a single thread and is not accessible by multiple threads.

The work area partition reference that is returned to a user implements javax.naming.Referenceable, as well as com.ibm.websphere.UserWorkArea, therefore a user can bind their partition into a name to make their partition publicly available. An alternative to using Java naming to bind and access the partition is to use the work area partition manager interface. Anyone can access the work area partition manager interface; therefore, if a user wants to make their partition publicly available, they simply need to publish their partition name. Other users can then call the getWorkAreaPartition method on the work area partition manager interface with the published name.

The WorkAreaPartitionManager.createWorkAreaPartition method can only be used from a Java 2 Platform, Enterprise Edition (J2EE) client. To create a work area partition on the server side, one must use the administrative console. On the server side a work area partition must be created during server startup because each partition needs to be register with the appropriate Web and Enterprise JavaBeans (EJB) collaborators before the server has started. Custom work area partitions are created by the work area partition service and defined by the UserWorkArea interface.

The work area partition service also allows a user to configure partitions with additional properties that are not available on the UserWorkArea partition, such as bidirectional propagation of work area partition context and deferred attribute serialization. These properties are available as configuration properties when creating a partition. For a complete list of the configuration properties available when creating a partition, please see the section ″Configurable Work Area Partition Properties″ in “The Work area partition manager interface” on page 2087. The properties are defined as follows:

Bidirectional propagation of work area context

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors are propagated to the target application. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information is propagated to any remote objects that it invokes.

Whether context changes propagate back to a calling application from a remote application depends on the configuration of the work area partition. If a user creates a partition to be bidirectional (selects the Bidirectional property during creation), changes made by a remote application propagates back to the calling application, meaning that changes made to the work area context by a downstream process will propagate back up stream. The UserWorkArea partition is not configured (and can never be configured) to

be bidirectional; therefore context changes only flow to downstream processes and do not propagate back upstream. See “Bidirectional propagation” on page 2092 for further explanation.

Deferred attribute serialization of work area context

By default, on each set operation the attribute set into a work area is automatically serialized by the work area service. On each subsequent get operation on that same attribute it is deserialized and returned to the requester. This gives the work area service complete control of the attribute such that any changes to a mutable object are not reflected in the work area's copy of the attribute unless a user specifically resets the attribute into the work area. However, this can potentially lead to excessive serialization and deserialization.

Excessive serialization and deserialization can result in observable performance degradation under heavy load. The deferred attribute serialization configuration property is a caching feature that reduces serialization and deserialization operations. When deferred attribute serialization is enabled in a client or server process, by selecting the Deferred Attribute Serialization field during the creation of the work area, attributes set into the work area service are not automatically serialized during the set operation. Rather, a reference to the attribute is stored in the work area. If the attribute is mutable, then changes to the object are reflected in the work area's reference to that attribute. When a get operation is performed on that attribute, the reference to that object is returned and no deserialization is performed.

Attributes are not actually serialized until the thread with which the attribute is associated makes a remote IIOP invocation. At that point the attribute is serialized and the serialized form of the attribute is cached. If the attribute is not reset into the work area, changes to the original attribute are still reflected within the attribute contained within the work area because the work area still holds a cached reference to the original object. However, if the work area has not been told that the attribute has changed by resetting the attribute into the work area, subsequent remote requests continues to use the cached serialized version of the attribute and direct changes to the mutable attribute are not propagated. This is an important distinction between enabling and not enabling the deferred attribute serialization configuration property and a user must pay close attention to this difference and how mutable objects are handled when enabling deferred attribute serialization. The work area service releases cached references and cached serialized versions of attributes when any of the following occur:

- An attribute is reset or removed.
- The work area is explicitly completed by the application.
- Server component ends execution of the request during which the work area was begun.
- Client process which began the work area terminates.

Partition context propagation across process boundaries

Work area context automatically propagates from client to server when a client makes a remote call to a server. If a client is configured with, for example, three different work area partitions when it makes a remote call to a server, server1; the context associated with each partition on the client thread propagates to server1. If the same three partitions reside (have been created) on server1, the context is demarshalled to the appropriate partition. However, if none or only a few of the three partitions have been created on server1, only the context associated with a partition that is resident on both the client and server is demarshalled. The context associated with a partition that is not resident on server1 is still resident on server1 but will not be accessible. The context associated with partitions that are not resident on server1 must remain resident on server1 in case another remote call is made to a different server. Going one step further, if server1 makes a call to yet another server, server2 and assume server2 has created all the same partitions that the client has, server2 receives the context for the partitions that were not resident on server1. Any partitions that reside on server1 that did not reside on the client, now have its context propagated to server2.

For additional information about work area, see the com.ibm.websphere.workarea package in the API documentation. The generated API documentation is available in the information center table of contents

from the path **Reference** > **Developer** > **API documentation** > **Application programming interfaces**.

The Work area partition manager interface

Applications interact with the work area partition service by using the work area partition manager interface. A user can retrieve an instance of the work area partition manager interface out of naming and use the methods that are defined in the following section.

An implementation of the work area partition manager interface is bound in Java naming at `java:comp/websphere/WorkAreaPartitionManager`. This interface is responsible for creating, retrieving, and manipulating work area partitions:

```
package com.ibm.websphere.workarea;

import com.ibm.websphere.workarea.UserWorkArea;
import com.ibm.websphere.workarea.PartitionAlreadyExistsException;
import com.ibm.websphere.workarea.NoSuchPartitionException;
import java.util.Properties;

public interface WorkAreaPartitionManager {

  //Returns an instance of a work area partition for the given name, or throws an exception if the
  //partition name doesn't exists.
  public UserWorkArea getWorkAreaPartition(String partitionName) throws NoSuchPartitionException;

  //Returns a new instance of a work area partition (an implementation of the UserWorkArea interface)
  //or throws an exception if the partition name already exists.  The createWorkAreaPartition should
  //only be used within a Java 2 platform, Enterprise Edition (J2EE) client and NOT on the
  //server.  To create a work area partition on the server, use the WebSphere administrative
  //console.
  public UserWorkArea createWorkAreaPartition(String partitionName, Properties props) throws
     PartitionAlreadyExistsException, java.lang.IllegalAccessException;
  }
}
```

EJB applications can use the work area partition manager interface only within the implementation of methods in either the remote or local interface, or both; likewise, servlets can use the interface only within the service method of the HTTPServlet class. Use of work areas within any life cycle method of a servlet or enterprise bean is considered a deviation from the work area programming model and is not supported.

Programmatically creating a work area partition through the createWorkAreaPartition method is only available on the Java 2 platform, Enterprise Edition (J2EE) client. To create a work area partition on the server, use the WebSphere administrative console as described in “Configuring work area partitions” on page 2084. All partitions in a server process must be created before server startup is complete so that the work area service can register with the appropriate container collaborators. Therefore, calling the createWorkAreaPartition method in a server process after the server starts results in a java.lang.IllegalAccessException exception. The createWorkAreaPartition method can be called in a J2EE application client at any time.

Configurable Work Area Partition Properties

This section applies to the use of the createWorkAreaPartition method on the WorkAreaPartitionManager interface. As is described above, this method should only be used on a Java 2 platform, Enterprise Edition (J2EE) client. To create a partition on the server, please see Configuring work area partitions.

The ″createWorkAreaPartition″ method on the WorkAreaPartitionManager interface takes a java.util.Properties objects. This Properties object, and the properties it contains, is used to define the work area partition. Below is an example of creating a Properties object and setting a property:

Note: A more detailed example of the usage of the WorkAreaPartitionManager can be found at “Example: Work area partition manager” on page 2089.

```
java.util.Properties props = new java.util.Properties():
props.put("maxSendSize","12345");
```

Acceptable key/values pairs (properties) for defining a partition are as follows:

- *maxSendSize* - Indicates the maximum size (bytes) of a work area that can be sent on a remote call. Acceptable values are:
  - ″-1″ = Uses the default size of 32767.
  - ″0″ = Unlimited size, this value will not be policed which might help performance a bit depending on the number of work area an application has.
  - ″1″ = Integer.MAX_VALUE
- *maxReceiveSize* - Indicates the maximum size (bytes) of a work area that can be received. Acceptable values are:
  - ″-1″ = Uses the default size of 32767.
  - ″0″ = Unlimited size, this value will not be policed which might help performance a bit depending on the number of work area an application has.
  - ″1″ = Integer.MAX_VALUE
- *Bidirectional* - Indicates if work area context that is changed by a downstream process should be propagated back upstream to the originator of that context. For a more complete description of this property, please see the section ″Bidirectional propagation of work area context″ at “Work area partition service” on page 2085. Acceptable values are:
  - ″true″ = Context changes will be returned from a remote call.
  - ″false″ = Context changes will not be returned from a remote call.

  Note: The default setting is ″false.″
- *DeferredAttributeSerialization* - Indicates if the serialization of attribute should be optimized to occur exactly once per process. For a more complete description of this property, please see the section ″Deferred attribute serialization of work area context″ at “Work area partition service” on page 2085. Acceptable values are:
  - ″true″
    - When an attribute is set into the work area, it will not be serialized until a remote request is made.
    - If the value is unchanged by response, the serialized form will be used for subsequent requests; the live object will be retrieved via getters.
    - When requests are made during a remote request, a value will be deserialized on demand exactly once. The serialized form will be used for subsequent requests from this remote process on this distributed thread; subsequent requests in process for the same attribute will return the already deserialized value. There are risks with concurrency with DeferredAttributeSerialization. After serialization in a client process, updates to the attribute will no longer be reflected in the work area’s copy until the value is explicitly reset through the UserWorkArea interface. Changes made to a retrieved reference in a downstream process will not be propagated to subsequent downstream requests (or returned on the reply as a changed value) unless explicitly reset through the UserWorkArea interface.
  - ″false″
    - When an attribute is set into the work area, it will be immediately serialized and the bytes will be stored.
    - When an attribute is retrieved from the work area, it will always be deserialized from stored bytes.

  Note: The default value is ″false.″
- *EnableWebServicePropagation* - Indicates if work area context should propagate on a WebService call. Acceptable values are:
  - ″true″ = Context will propagate on a WebService call.
  - ″false″ = Context will not propagate on a WebService call.

Note: The default value is ″false.″

Exceptions

The work area partition service defines the following exceptions for use with the work area partition manager interface:

PartitionAlreadyExistsException
This exception is raised by the createWorkAreaPartition method on the WorkAreaPartitionManager implementation if a user tries to create a work area partition with a partition name that already exists. Partition names must be unique.

NoSuchPartitionException
This exception is raised by the getWorkAreaPartition method on the WorkAreaPartitionManager implementation if a user requests a work area partition with a partition name that does not exist.

java.lang.IllegalAccessException
This exception is raised by the createWorkAreaPartition method on the WorkAreaPartitionManager implementation if a user tries to create a work area partition during run time on a server process. This method can only be used on a J2EE client process. In the server process, a partition must be created using the administrative console.

For additional information about work area, see the com.ibm.websphere.workarea package in the API documentation. The generated API documentation is available in the information center table of contents from the path **Reference** > **Developer** > **API documentation** > **Application programming interfaces**.

Example: Work area partition manager

The example below demonstrates the use of the work area partition manager interface. The sample illustrates how to create and retrieve a work area partition programmatically`. Please note that programmatically creating a work area partition is only available on the Java 2 platform, Enterprise Edition (J2EE) client. To create a work area partition on the server one must use the administrative console. See “Work area partition service” on page 2085 for configuration parameters available to configure a partition.

```
import com.ibm.websphere.workarea.WorkAreaPartitionManager;
import com.ibm.websphere.workarea.UserWorkArea;
import com.ibm.websphere.workarea.PartitionAlreadyExistsException;
import com.ibm.websphere.workarea.NoSuchPartitionException;
import java.lang.IllegalAccessError;
import java.util.Properties;
import javax.naming.InitialContext;

//This sample demonstrates how to retrieve an instance of the
//WorkAreaPartitionManager implementation and how to use that
//instance to create a WorkArea partition and retrieve a partition.
//NOTE: Creating a partition in the way listed below is only available
//on a J2EE client.  To create a partition on the server use the
//WebSphere administrative console.  Retrieving a WorkArea
//partition is performed in the same way on both client and server.

public class Example {

     //The name of the partition to create/retrieve
     String partitionName = "myPartitionName";
     //The name in java naming the WorkAreaPartitionManager instance is bound to
     String jndiName = "java:comp/websphere/WorkAreaPartitionManager";

     //On a J2EE client a user would create a partition as follows:
     public UserWorkArea myCreate(){
         //Variable to hold our WorkAreaPartitionManager reference
         WorkAreaPartitionManager partitionManager = null;
         //Get an instance of the WorkAreaPartitionManager implementation
         try {
             InitialContext initialContext = new InitialContext();
             partitionManager = (WorkAreaPartitionManager) initialContext.lookup(jndiName);
```

```
        } catch (Exception e) {  }

        //Set the properties to configure our WorkArea partition
        Properties props = new Properties();
        props.put("maxSendSize","12345");
        props.put("maxReceiveSize","54321");
        props.put("Bidirectional","true");
     props.put("DeferredAttributeSerialization","true");

        //Variable used to hold the newly created WorkArea Partition
        UserWorkArea myPartition = null;

        try{
            //This is the way to create a partition on the J2EE client.  Use the
            //WebSphere Administrative Console to create a WorkArea Partition
            //on the server.
            myPartition = partitionManager.createWorkAreaPartition(partitionName,props);
        }
        catch (PartitionAlreadyExistsException e){  }
        catch (IllegalAccessException e){  }

        return myPartition;
    }

    //. . . .

    //In order to retrieve a WorkArea partition at some time later or
    //from some other class, do the following (from client or server):
    public UserWorkArea myGet(){
        //Variable to hold our WorkAreaPartitionManager reference
        WorkAreaPartitionManager partitionManager = null;
        //Get an instance of the WorkAreaPartitionManager implementation
        try {
            InitialContext initialContext = new InitialContext();
            partitionManager = (WorkAreaPartitionManager) initialContext.lookup(jndiName);
        } catch (Exception e) {  }

        //Variable used to hold the retrieved WorkArea partition
        UserWorkArea myPartition = null;
        try{
            myPartition = partitionManager.getWorkAreaPartition(partitionName);
        }catch(NoSuchPartitionException e){  }

        return myPartition;
    }
}
```

Work area partition collection

Use this page to manage the work area service.

The work area partition service supports the definition of custom work area partitions.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Business Process Services > Work area partition service**.

Name:

Specifies the name of the work area partition that is used to retrieve the partition. This name must be unique.

Description:

Specifies the description of the work area partition.

Enable service at server startup:

Specifies whether the server attempts to start the specified service when the server starts.

Bidirectional:

Permits applications to modify the context of a work area that is imported by a J2EE request; modified properties are propagated back to the requestor environment. This option is disabled by default.

Maximum send size:

Specifies the maximum size of data that can be sent within a single work area. (0 = no limit; -1 = default)

A value of 0 means there is no limit to the data sent. The default value of -1 represents 32768 bytes of data sent.

Maximum receive size:

Specifies the maximum size of data that can be received within a single work area. (0 = no limit; -1 = default)

A value of 0 means there is no limit to the data received. The default value of -1 represents 32768 bytes of data received.

Deferred attribute serialization:

Specifies whether attribute serialization is deferred until the work area is propagated on a remote invocation.

Enable Web service propagation:

Specifies whether the work area partition is propagated on Web service requests. This option is disabled by default.

Work area partition settings:

Use this page to modify the work area partition settings.

The work area partition service supports the definition of custom work area partitions.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Business Process Services > Work area partition service >** *work_area_partition_name*.

Name:

Specifies the name of the work area partition that is used to retrieve the partition. This name must be unique.

Description:

Specifies the description of the work area partition.

Enable service at server startup:

Specifies whether the server attempts to start the specified service when the server starts.

Bidirectional:

Permits applications to modify the context of a work area that is imported by a J2EE request; modified properties are propagated back to the requestor environment. This option is disabled by default.

Maximum send size:

Specifies the maximum size of data that can be sent within a single work area. (0 = no limit; -1 = default)

Data type	Integer
Units	Bytes
Default	32768
Range	-1, 0 (no limit) and 1 to 2147483647

Maximum receive size:

Specifies the maximum size of data that can be received within a single work area. (0 = no limit; -1 = default)

Data type	Integer
Units	Bytes
Default	32768
Range	-1, 0 (no limit) and 1 to 2147483647

Deferred attribute serialization:

Specifies whether attribute serialization is deferred until the work area is propagated on a remote invocation. This option is disabled by default.

Enable Web service propagation:

Specifies whether the work area partition is propagated on Web service requests. This option is disabled by default.

Bidirectional propagation

Example: Bidirectional propagation of work area context

Whether context changes propagate back to a calling application from a remote application depends on the configuration of the work area partition. If a user creates a bidirectional partition, changes made by a remote application propagate back to the calling application. In other words, changes made to the work area context by a downstream process propagate back up stream. Figure 1 illustrates distribution of work area context on a remote call when the partition containing the given work area is configured for bidirectional propagation of its work area context. For this illustration, the client and server must have created a partition with the same name.

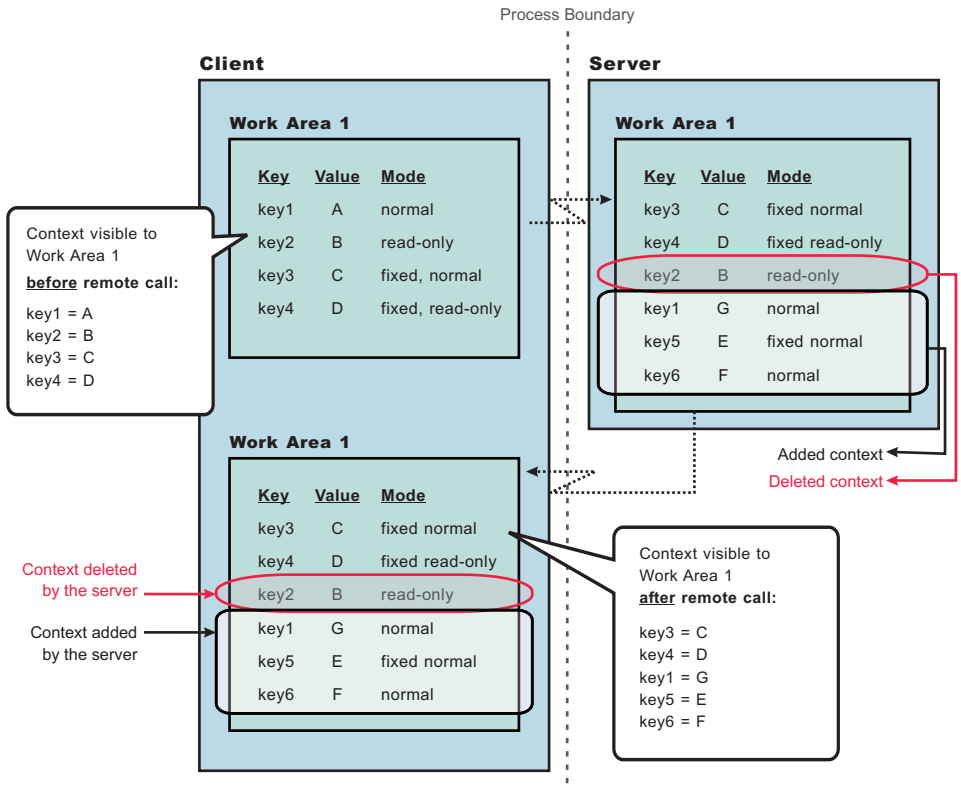


Figure 23. Figure 1

As Figure 1 shows, when the client makes a remote call to the server, the server receives the context set by the client process. The server then can make changes to this context or add to it. In this illustration, the server overwrites the value at **key1**, removes the property at **key2**, and adds two new properties at **key5** and **key6**. When the server application returns to the client, the work area context is propagated back to the client and demarshalled. The current work area is then updated with the new context. Note, that if the partition is not configured as bidirectional, and the server tries to change or remove context in work area, ″Work Area 1″, it will receive a com.ibm.websphere.workarea.NotOriginator exception since the client was the originator of the work area. The server can retrieve the context in ″Work Area 1″. This is the main distinction between bidirectional propagation of context and non-bidirectional propagation.

Bidirectional propagation of nested work area context

If a remote application needs to add context to a work area that is only used by itself or any other remote objects, the remote application should begin another work area. By beginning a new work area, the new context added is scoped to that application and does not flow back to the calling application. The major benefit of nesting work areas is that nesting work areas allows an application to scope work area context to a given application. Taking the above illustration one step further, if the server has begun a work area before overwriting the value at **key1**, removing the property at **key2**, or adding new properties at **key5** and **key6**; those changes would not have propagated back to the client. This is shown in Figure 2. You can also see from this figure that the client does not receive the context from the nested work area started by the server.

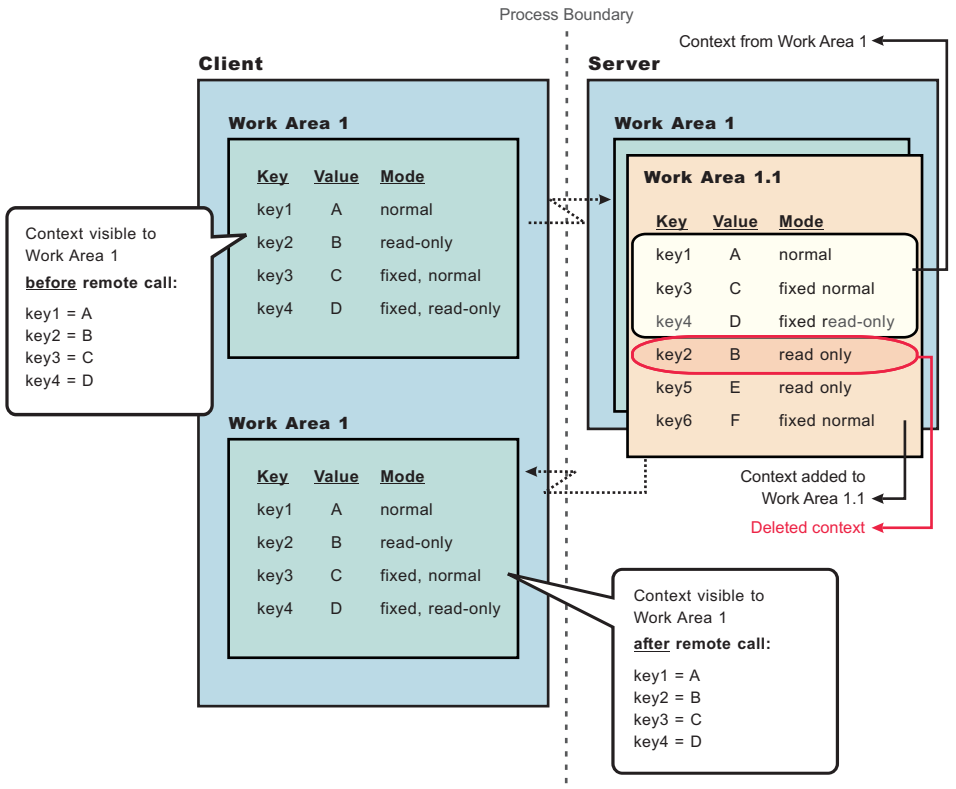


Figure 24. Figure 2

Accessing a user defined work area partition

The work area partition service provides a Java Naming and Directory Interface (JNDI) binding to an implementation of the work area partition manager interface under the name java:comp/websphere/WorkAreaPartitionManager. Applications that need to access their partition can perform a lookup on that JNDI name and then use the getWorkAreaPartition method on the work area partition manager, as shown in the following code example:

```
import com.ibm.websphere.workarea.*;
import javax.naming.*;

public class SimpleSampleServlet {
  ...

        //Variable to hold our WorkAreaPartitionManager implementation
        WorkAreaPartitionManager partitionManager = null;
        try {
            InitialContext initialContext = new InitialContext();
            partitionManager = (WorkAreaPartitionManager)
           initialContext.lookup("java:comp/websphere/WorkAreaPartitionManager");
        } catch (Exception e) {...}

        //Variable used to hold the retrieved WorkArea Partition
        UserWorkArea myPartition = null;
        try{
            myPartition = partitionManager.getWorkAreaPartition(partitionName);
        }catch(NoSuchPartitionException e){...}
}
```

The next step is to use the begin method to create a new work area and associate it with the calling thread, as described in the topic Beginning a new work area.

Propagating work area context over Web services

WebSphere Application Server Version 6.1 introduces the option to propagate work area context on a Web service call. Prior to WebSphere Application Server Version 6.1, work area context was only propagated over RMI/IIOP calls. The work area application programming interfaces (APIs) have not changed to implement this propagation. You can use the work area APIs as they have in the past and as outlined in the work area documentation. However, by default, work area context is not propagated on a Web service call, you must enable this option.

1. Enable a server to propagate work area context on a Web service call.
   a. Start the administrative console.
   b. Select **Servers > Application servers > *server_name* > Business Process Services**.
      - To enable the work area service, (the UserWorkArea partition) to propagate its context on a Web service call:
        – Select **Work area service**.
      - To enable an individual partition to propagate its context on a Web service call:
        – Select **Work area partition service**.
        – Select a partition.
   c. Check the EnableWebServicePropagation field to enable Web service propagation.
   d. Save the new configuration and restart the server to apply the new configuration.
2. Enable a client to propagate work area context on a Web service call:

   Note: The steps below are for the work area service (the UserWorkArea partition). For user defined partitions the EnableWebServicePropagation property must be set when creating a partition on the client, see “The Work area partition manager interface” on page 2087.

   a. Set the property com.ibm.websphere.workarea.EnableWebServicePropagation to true when invoking the launchClient script found in the $WAS_HOME/bin directory. For example, to set this property to true, add the following system properties to the launchClient invocation as needed: -CCDcom.ibm.websphere.workarea.EnableWebServicePropagation=true
   b. Set the property com.ibm.websphere.workarea.EnableWebServicePropagation in a property file that is used by the launchClient script. See “Running application clients” on page 242for additional information.

Chapter 21. Overview and new features for monitoring

Use the links provided in this topic to learn about monitoring capabilities.

New for administrators: Improved monitoring and performance tuning
A section of this topic describes what is new in the area of performance tuning.

Performance: Resources for learning

Use the following links to find relevant supplemental information about performance. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas. The following sections are covered in this reference:

View the following links for additional information:

- "Request metrics "
- "Monitoring performance with third-party tools"
- "Tuning performance"
- "Java™ performance resource" on page 2098

Request metrics

- Systems Management: Application Response Measurement (ARM)

  The Open Group ARM specifications.

Monitoring performance with third-party tools

- IBM Search Solutions.

  Use IBM's Global Solution Directory to find a list of IBM's business partners that offer performance monitoring tools compliant with WebSphere Application Server.

Tuning performance

- Hints on Running a high-performance Web server

  Read hints about running Apache on a heavily loaded Web server. The suggestions include how to tune your kernel for the heavier TCP/IP load, and hardware and software conflicts.
- Application tuning

  See WebSphere Application Server Development Best Practices for Performance and Scalability for more information on application tuning.
- Performance Analysis for Java Web sites

  Offers clear explanations and expert practical guidance on performance analysis for Java-based Web sites. It offers extensive appendices, including worksheets for capacity planning, checklists to help you prepare for different stages of performance testing, and a list of performance-test tool vendors.
- AIX AIX documentation

  View the entire AIX software documentation library for releases 4.3, 5.1, and 5.2.
- WebSphere Application Server Development Best Practices for Performance and Scalability

  Describes development best practices for Web applications with servlets, JavaServer Pages files, JDBC connections, and enterprise applications with Enterprise JavaBeans components.
- IBM WebSphere Application Server Advanced Edition Tuning Guide (Version 4.02)
- Redbook: WebSphere Application Server V3.5 Handbook (SG24-6161-00)
- Redbook: WebSphere Application Server V3 Performance Tuning Guide (SG24-5657-00)

Java™ performance resource

- IBM developerWorks

  Search the IBM developerWorks Web site for a list of garbage collection documentation, including ″Understanding the IBM Java Garbage Collector″, a three-part series. To locate the documentation, search on ″sensible garbage collection″ in the developerWorks search application.

  Review ″Understanding the IBM Java Garbage Collector″ for a description of the IBM verbose:gc output and more information about the IBM garbage collector.

Chapter 22. How do I monitor?

Hold your cursor over the task icon () to see a description of the task. The task preview feature is unavailable for Mozilla Web browsers.

Enable and customize PMI data collection	Documentation
Monitor system health by viewing PMI data	Documentation
Measure application flow	Documentation

Legend for ″How do I?...″ links

Detailed steps	Show me	Tell me	Guide me	Teach me
Refer to the detailed steps and reference	Watch a brief multimedia demonstration	View the presentation for an overview	Be led through the console pages	Perform the tutorial with sample code
Approximate time: Varies	**Approximate time:** 3 to 5 minutes	**Approximate time:** 10 minutes+	**Approximate time:** 1/2 hour+	**Approximate time:** 1 hour+

Chapter 23. Monitoring end user response time

To analyze Web site response from a client viewpoint, use Tivoli Monitoring for Transaction Performance.

Monitoring system response time provides an external perspective of how the overall Web site responds and performs from an client viewpoint. It is important to understand the load and response time on your site. You have some options when monitoring at this level.

- Use Tivoli Monitoring for Transaction Performance to support you to inject and monitor synthetic transactions, helping you identify when your Web site experiences a problem.
- You may also use other performance monitoring tools.

Chapter 24. Monitoring overall system health

Monitoring overall system health is fundamentally important to understanding the health of every system involved with your system. This includes Web servers, application servers, databases, back-end systems, and any other systems critical to running your Web site.

If any system has a problem, it might cause the `servlet is slow` message to appear. IBM and several other business partners leverage the WebSphere APIs to capture performance data and to incorporate it into an overall 24-by-7 monitoring solution. WebSphere Application Server provides Performance Monitoring Infrastructure (PMI) data to help monitor the overall health of the WebSphere Application Server environment. PMI provides average statistics on WebSphere Application Server resources, application resources, and system metrics. Many statistics are available in WebSphere Application Server, and you might want to understand the ones that most directly measure your site's resources to detect problems.

To monitor overall system health, monitor the following statistics at a minimum:

Metric	Meaning
Average response time	Include statistics, for example, servlet or enterprise beans response time. Response time statistics indicate how much time is spent in various parts of WebSphere Application Server and might quickly indicate where the problem is (for example, the servlet or the enterprise beans).
Number of requests (transactions)	Enables you to look at how much traffic is processed by WebSphere Application Server, helping you to determine the capacity that you have to manage. As the number of transactions increase, the response time of your system might be increasing, showing the need for more system resources or the need to retune your system to handle increased traffic.
Number of live HTTP sessions	The number of live HTTP sessions reflects the concurrent usage of your site. The more concurrent live sessions, the more memory is required. As the number of live sessions increase, you might adjust the session time-out values or the Java virtual machine (JVM) heap available.
Web server thread pools	Interpret the Web server thread pools, the Web container thread pools, and the Object Request Broker (ORB) thread pools, and the data source or connection pool size together. These thread pools might constrain performance due to their size. The thread pools setting can be too small or too large, therefore causing performance problems. Setting the thread pools too large impacts the amount of memory that is needed on a system or might cause too much work to flow downstream if downstream resources cannot handle a high influx of work. Setting thread pools too small might also cause bottlenecks if the downstream resource can handle an increase in workload.
The Web and Enterprise JavaBeans (EJB) thread pools	
Database and connection pool size	
Java virtual memory (JVM)	Use the JVM metric to understand the JVM heap dynamics, including the frequency of garbage collection. This data can assist in setting the optimal heap size. In addition, use the metric to identify potential memory leaks.
CPU	You must observe these system resources to ensure that you have enough system resources, for example, CPU, I/O, and paging, to handle the workload capacity.
I/O	
System paging	

To monitor several of these statistics, WebSphere Application Server provides the Performance Monitoring Infrastructure to obtain the data, and provides the Tivoli Performance Viewer (TPV) in the administrative console to view this data.

1. Enable PMI through the administrative console to begin data collection.
2. Use TPV or third-party performance monitoring and management solutions to monitor performance.
3. Extend monitoring capabilities by developing your own monitoring applications or extending PMI.

Performance Monitoring Infrastructure (PMI)

Use this page to learn about Performance Monitoring Infrastructure and other tools to help monitor the overall health of the application server.

A typical Web system consists of a Web server, application server, and a database. Monitoring and tuning the application server is critical to the overall performance of the Web system. Performance Monitoring Infrastructure (PMI) is the core monitoring infrastructure for WebSphere Application Server and WebSphere family products like, Portal, Commerce, and so on. The performance data provided by WebSphere PMI helps to monitor and tune the application server performance.

When tuning the WebSphere Application Server for optimal performance, or fixing a poorly performing Java 2 Platform, Enterprise Edition (J2EE) application, it is important to understand how the various run time and application resources are behaving from a performance perspective. PMI provides a comprehensive set of data that explains the runtime and application resource behavior. For example, PMI provides database connection pool size, servlet response time, Enterprise JavaBeans (EJB) method response time, Java virtual machine (JVM) garbage collection time, CPU usage, and so on. This data can be used to understand the runtime resource utilization patterns of the thread pool, connection pool, and so on, and the performance characteristics of the application components like servlets, JavaServer Pages (JSP), and enterprise beans.

Using PMI data, the performance bottlenecks in the application server can be identified and fixed. For instance, one of the PMI statistics in the Java DataBase Connectivity (JDBC) connection pool is the *number of statements discarded from prepared statement cache*. This statistic can be used to adjust the prepared statement cache size in order to minimize the discards and to improve the database query performance. PMI data can be monitored and analyzed by Tivoli Performance Viewer (TPV), other Tivoli tools, your own applications, or third party tools. TPV is a graphical viewer for PMI data that ships with WebSphere Application Server. Performance advisors use PMI data to analyze the run-time state of the application server, and provide tuning advice to optimize the application server resource utilization.

PMI data can also be used to monitor the health of the application server. Some of the health indicators are CPU usage, Servlet response time, and JDBC query time. Performance management tools like Tivoli Monitoring for Web Infrastructure and other third party tools can monitor the PMI data and generate alerts based on some predefined thresholds.

PMI architecture

The Performance Monitoring Infrastructure (PMI) uses a client-server architecture.

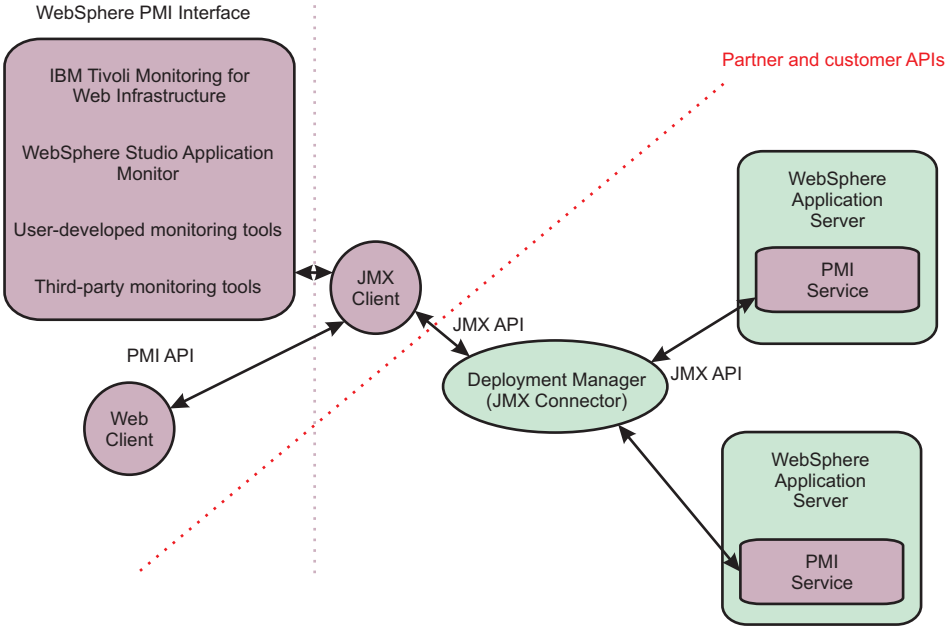
The server collects performance data from various WebSphere Application Server components. A client retrieves performance data from one or more servers and processes the data. WebSphere Application Server, Version 6 supports the Java™ 2 Platform, Enterprise Edition (J2EE) Management Reference Implementation (JSR-77).

In WebSphere Application Server, Version 6 and later, PMI counters are enabled, based on a monitoring or instrumentation level. The levels are None, Basic, Extended, All, and Custom. These levels are specified in the PMI module XML file. Enabling the module at a given level includes all the counters at the given level plus counters from levels below the given level. So, enabling the module at the extended level enables all the counters at that level plus all the basic level counters as well.

JSR-077 defines a set of statistics for J2EE components as part of the StatisticProvider interface. The PMI monitoring level of Basic includes all of the JSR-077 specified statistics. PMI is set to monitor at a *Basic* level by default.

As shown in the following figure, the server collects PMI data in memory. This data consists of counters such as servlet response time and data connection pool usage. The data points are then retrieved using a

Web client, a Java client, or a Java Management Extensions (JMX) client. WebSphere Application Server contains Tivoli Performance Viewer, a Java client which displays and monitors performance data. See the “Monitoring performance with Tivoli Performance Viewer (TPV)” on page 2258, “Third-party performance monitoring and management solutions” on page 2273, and “Developing your own monitoring applications” on page 2223 topics for more information on monitoring tools.



The figure shows the overall PMI architecture. On the right side, the server updates and keeps PMI data in memory. The left side displays a Web client, a Java client, and a JMX client retrieving the performance data.

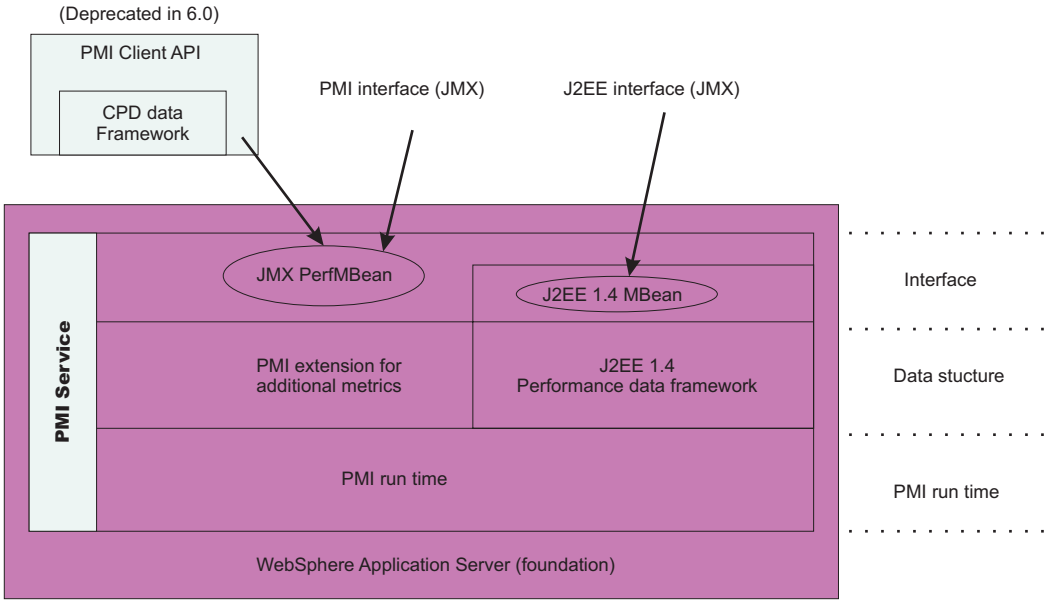
PMI and J2EE 1.4 Performance Data Framework

Java 2 Platform, Enterprise Edition (J2EE) 1.4 includes a Performance Data Framework that is defined as part of JSR-077 (Java 2 Platform, Enterprise Edition Management Specification).

This framework specifies the performance data that must be available for various J2EE components. WebSphere PMI complies with J2EE 1.4 standards by implementing the J2EE 1.4 Performance Data Framework.

In addition to providing statistics that are defined in J2EE 1.4, PMI provides additional statistics about the J2EE components, for example, servlets and enterprise beans, and WebSphere Application Server-specific components, for example, thread pools and workload management.

The following diagram shows how the PMI and J2EE Performance Data Framework fit into WebSphere Application Server.



PMI data classification

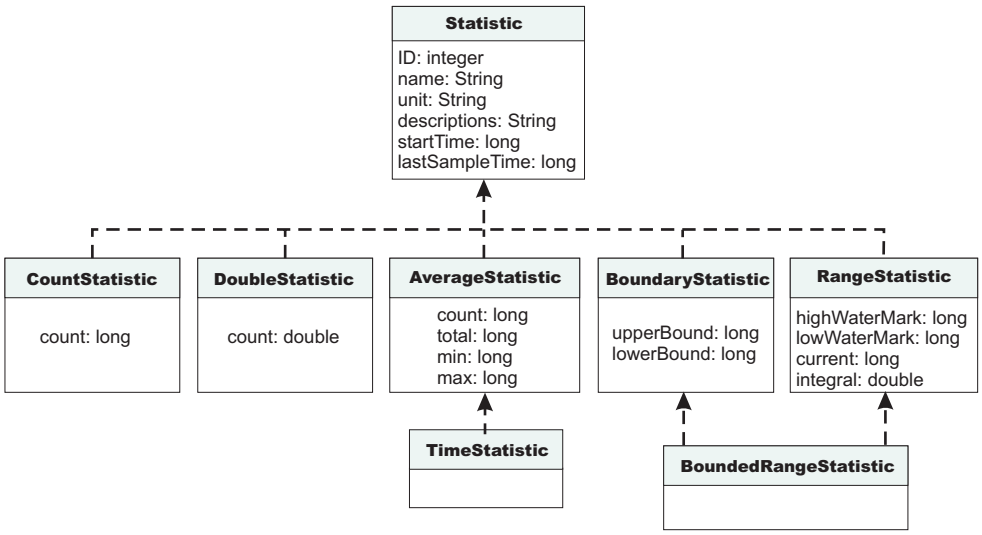
This topic describes the Performance Monitoring Infrastructure (PMI) data classification.

The static component consists of a name, ID and other descriptive attributes to identify the data. The dynamic component contains information that changes over time, such as the current value of a counter and the time stamp associated with that value.

The PMI data can be one of the following statistic types (these statistic types follow the J2EE 1.4 Performance Data Framework):

Statistic type	Description	Example
CountStatistic	Represents a running count of a given value.	Number of Servlet requests
AverageStatistic	Represents a simple average. Keeps track of total, count, min, and max. The average can be derived by total and count. (This type is WebSphere extension to J2EE Performance Data Framework)	Average HttpSession size in bytes.
TimeStatistic	Same as AverageStatistic, except that the unit of measure is milliseconds or seconds.	Average Servlet response time.
RangeStatistic	Represents a time-weighted average. Keeps track of current, low water mark, high water mark, time-weight total, and integral.	Number of concurrent Servlet requests.
BoundedRangeStatistic	Same as RangeStatistic, with lower bound and upper bound.	JDBC connection pool size.

The following diagram shows the statistic class hierarchy:



Statistic

ID A unique ID that identifies the Statistic within the given Stats (WebSphere PMI extension)

name Statistic name

unit Unit of measurement for the statistic

description
Textual description of the statistic

startTime
Time the first measurement was taken

lastSampleTime
Time the most recent measurement was taken

CountStatistic

count Count since the measurement started

DoubleStatistic

count Value since the measurement started

AverageStatistic
(WebSphere PMI extension. This is the same as the TimeStatistic defined in J2EE 1.4, except that it is used to track non-time-related measurements like byte size, etc.)

count Number of measurements

total Sum of the values of all the measurements

min Minimum value

max Maximum value

BoundaryStatistic

upperBound
Upper limit of this attribute

lowerBound
Lower limit of this attribute

RangeStatistic

current
Current value of this attribute

lowWaterMark
Lowest value of this attribute

upperWaterMark
Highest value of this attribute

integral
Time-weighted sum of this attribute [time-weighted average = integral / (lastSampleTime - startTime] (WebSphere PMI extension)

In WebSphere Application Server, Version 4, PMI data was classified with the following types:
- **Numeric**: Maps to CountStatistic in the J2EE 1.4 specification. Holds a single numeric value that can either be a long or a double. This data type is used to keep track of simple numeric data, such as counts.
- **Stat**: Holds statistical data on a sample space, including the number of elements in the sample set, their sum, and sum of squares. You can obtain the mean, variance, and standard deviation of the mean from this data.
- **Load**: Maps to the RangeStatistic or BoundedRangeStatistic, based on J2EE 1.4 specification. This data type keeps track of a level as a function of time, including the current level, the time that level was reached, and the integral of that level over time. From this data, you can obtain the time-weighted average of that level. For example, this data type is used in the number of active threads and the number of waiters in a queue.

These PMI data types continue to be supported through the PMI client API. Statistical data types are supported through both the PMI API and Java Management Extension (JMX) API.

In WebSphere Application Server, Version 4 and Version 5, CountStatistic data require a *low* monitoring level, and TimeStatistic data require a *medium* monitoring level. RangeStatistic and BoundedRangeStatistic require a *high* monitoring level. There are a few counters that are exceptions to this rule. The average method response time, the total method calls, and active methods counters require a *high* monitoring level. The Java virtual machine counters, SerializableSessObjSize, and data tracked for each individual method (method level data) require a *maximum* monitoring level. Also, the level *maximum* enables synchronized update to all the statistic types.

WebSphere Application Server Versions 6.0 and above deprecate the monitoring levels (*Low*, *Medium*, *High*, and *Max*) and introduces fine-grained control to enable/disable statistics individually. The fine-grained control is available under the custom option. Refer to “Enabling PMI using the administrative console” on page 2213 for more details.

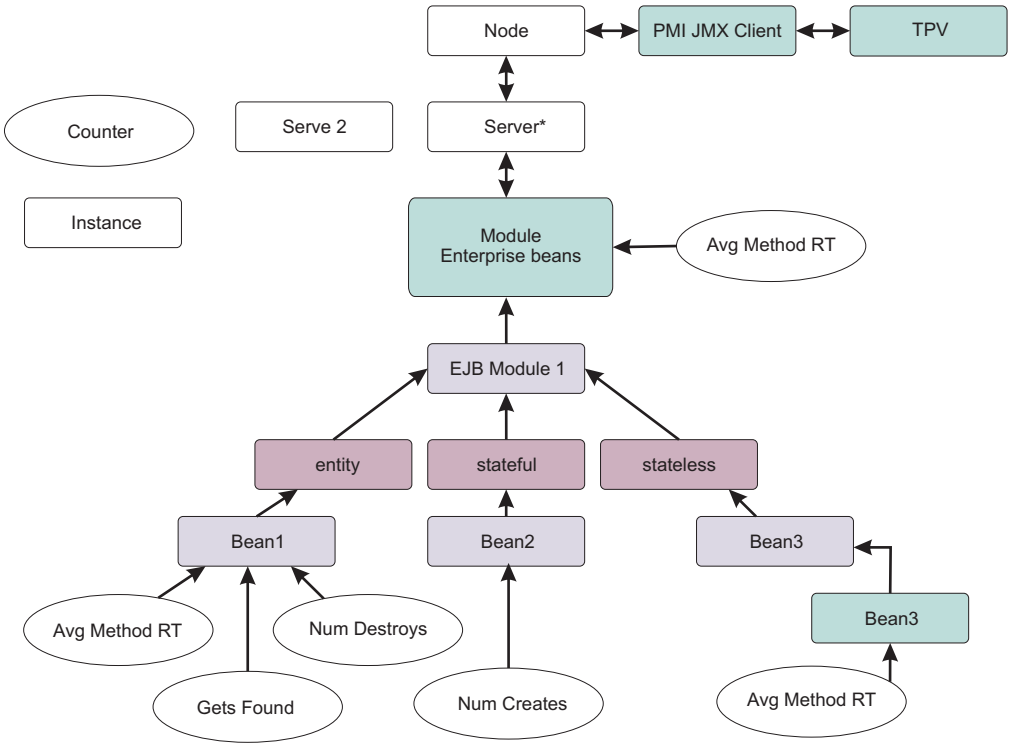
In order to reduce the monitoring overhead, updates to CountStatistic, DoubleStatistic, AverageStatistic, and TimeStatistic are not synchronized. Since this data tracks the total and average, the extra accuracy is generally not worth the performance cost. RangeStatistic and BoundedRangeStatistic are very sensitive; therefore, they are always synchronized. To enable synchronized updates for all the statistic types enable the ’Use sequential update’ option. Refer to “Enabling PMI using the administrative console” on page 2213 for details.

PMI data organization

Use this page as a general overview of monitoring, data collection, and counters using Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV).

Performance Monitoring Infrastructure (PMI) provides server-side monitoring and a client-side API to retrieve performance data. PMI maintains statistical data within the entire WebSphere Application Server domain, including multiple nodes and servers. Each node can contain one or more WebSphere Application Server. Each server organizes PMI data into modules and submodules.

Hierarchy of data collections used for performance reporting to Tivoli Performance Viewer (TPV)



Tivoli Performance Viewer, formerly Resource Analyzer, organizes performance data in a centralized hierarchy of the following objects:

- **Node.** A node represents a physical machine in the WebSphere Application Server administrative domain.
- **Server.** A server is a functional unit that provides services to clients over a network. No performance data is collected for the server itself.
- **Module.**

  A module represents one of the resource categories for which collected data is reported to the performance viewer. Each module has at least one configuration file in XML format. These files determine organization and lists a unique identifier for each performance data in the module. Modules include enterprise beans, JDBC connection pools, J2C connection pool, Java virtual machine (JVM) run time (including Java Virtual Machine Tool Interface (JVMTI)), servlet session manager, thread pools, transaction manager, Web applications, Object Request Broker (ORB), Workload Management (WLM), Web services gateway (WSGW), and dynamic cache.
- **Submodule.** A submodule represents a fine granularity of a resource category under the module. For example, ORB thread pool is a submodule of the thread pool category. Submodules can contain other submodules.
- **Counter.** A counter is a data type used to hold performance information for analysis. Each resource category (module) has an associated set of counters. The data points within a module are queried and distinguished by the MBean ObjectNames or PerfDescriptors. Examples of counters include the number of active enterprise beans, the time spent responding to a servlet request and the number of kilobytes of available memory.

Tivoli Performance Viewer is a thin client integrated into the WebSphere Application Server administrative console. It provides a simple viewer for the performance data provided by Performance Monitoring Infrastructure (PMI), and allows users to view and manipulate the data for counters. A particular counter type can appear in several modules. For example, both the servlet and enterprise bean modules have a

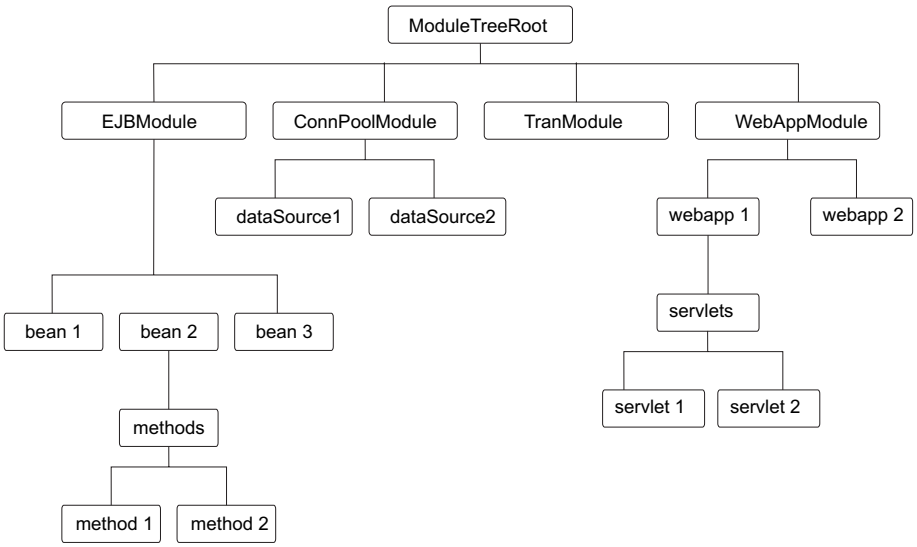
response time counter. In addition, a counter type can have multiple instances within a module. In the figure above, both the Enterprise beans module and Bean1 have an Avg Method RT counter.

Counters are enabled at the module level and can be enabled or disabled for elements within the module. For example, in the figure, if the enterprise beans module is enabled, its Avg Method RT counter is enabled by default. However, you can then disable the Avg Method RT counter even when the rest of the module counters are enabled. You can also, if desired, disable the Avg Method RT counter for Bean1, but the aggregate response time reported for the whole module no longer includes Bean1 data.

As part of a fine-grained control feature, WebSphere Application Server provides statistic sets which are pre-defined, fixed server-side sets, based on the PMI statistic usage scenarios. The PMI specification levels include: *none*, *basic*, *extended*, *all*, or *custom*. If you choose *none*, all PMI modules are disabled. Choosing *basic* provides the J2EE and the essential set of statistics to give you a basic level of monitoring. Selecting *extended* gives you the basic level of monitoring plus Work Load Monitor, Performance Advisor, and Tivoli resource models for a more robust monitoring set. Choosing *all* enables all statistics. Choosing *custom* gives you fine-grained control to enable or disable statistics individually.

There are only two states for a statistic: enabled or disabled. To provide an option to enable synchronized updates, WebSphere Application Server, provides a configuration parameter, synchronizedUpdate, at the PMI service level. When this attribute is true, all the statistic updates are synchronized. By default, the synchronizedUpdate parameter is set to false. The synchronizedUpdate parameter is the equivalent to the Max level in V5.0.*x* and V5.1*x*.

Data collection can affect performance of the application server. The impact depends on the number of counters enabled, the type of counters enabled and the monitoring level set for the counters.



The following PMI modules are available to provide statistical data:

Enterprise bean module, enterprise bean, methods in a bean

Data counters for this category report load values, response times, and life cycle activities for enterprise beans. Examples include the average number of active beans and the number of times bean data is loaded or written to the database. Information is provided for enterprise bean methods and the remote interfaces used by an enterprise bean. Examples include the number of times a method is called and the average response time for the method. In addition, the Tivoli Performance Viewer reports information on the size and use of a bean objects cache or enterprise bean object pool. Examples include the number of calls attempting to retrieve an object from a pool and the number of times an object is found available in the pool.

JDBC connection pools
Data counters for this category contain usage information about the JDBC connection pools for a database. Examples include the number of managed connections or physical connections and the total number of connections or connection handles.

Java 2 Connector (J2C) connection pool
Data counters for this category contain usage information about the Java 2 Platform, Enterprise Edition (J2EE) Connector architecture that enables enterprise beans to connect and interact with procedural back-end systems, such as Customer Information Control System (CICS), and Information Management System (IMS). Examples include the number of managed connections or physical connections and the total number of connections or connection handles.

Java virtual machine API (JVM)
Data counters for this category contain memory used by a process as reported by JVM run time. Examples are the total memory available and the amount of free memory for the JVM. JVM run time also includes data from the JVMTI. This data provides detailed information about the JVM running the application server.

Servlet session manager
Data counters for this category contain usage information for HTTP sessions. Examples include the total number of accessed sessions, the average amount of time it takes for a session to perform a request, and the average number of concurrently active HTTP sessions.

Thread pool
Data counters for this category contain information about the thread pools for Object Request Broker (ORB) threads and the Web container pools used to process HTTP requests. Examples include the number of threads created and destroyed, the maximum number of pooled threads allowed, and the average number of active threads in the pool.

Java Transaction API (JTA)
Data counters for this category contain performance information for the transaction manager. Examples include the average number of active transactions, the average duration of transactions, and the average number of methods per transaction.

Web applications, servlet
Data counters for this category contain information for the selected server. Examples include the number of loaded servlets, the average response time for completed requests, and the number of requests for the servlet.

Object Request Broker (ORB)
Data counters for this category contain information for the ORB. Examples include the object reference lookup time, the total number of requests, and the processing time for each interceptor.

Web services gateway (WSGW)
Data counters for this category contain information for WSGW. Examples include the number of synchronous and asynchronous requests and responses.

System data
Data counters for this category contain information for a machine (node). Examples include the CPU utilization and memory usage. Note that this category is available at node level, which means it is only available at node agent in the multiple servers version.

Workload Management (WLM)
Data counters for this category contain information for workload management. Examples include the number of requests, the number of updates and average response time.

Dynamic cache
Data counters for this category contain information for the dynamic cache service. Examples include in-memory cache size, the number of invalidations, and the number of hits and misses.

Web services
Data counters for this category contain information for the Web services. Examples include the number of loaded Web services, the number of requests delivered and processed, the request response time, and the average size of requests.

Alarm manager
Data counters for this category contain information for the Alarm Manager.

Object pool
Data counters for this category contain information for Object Pools.

Scheduler
Data counters for this category contain information for the Scheduler service.

You can access PMI data through the getStatsObject and the getStatsArray method in the PerfMBean. You need to pass the MBean ObjectName(s) to the PerfMBean.

Use the following MBean types to get PMI data in the related categories:

- DynaCache: dynamic cache PMI data
- EJBModule*: Enterprise Java Bean (EJB) module PMI data (BeanModule)
- EntityBean*: specific EJB PMI data (BeanModule)
- JDBCProvider*: JDBC connection pool PMI data
- J2CResourceAdapter*: Java 2 Connectivity (J2C) connection pool PMI data
- JVM: Java virtual machine PMI data
- MessageDrivenBean*: specific EJB PMI data (BeanModule)
- ORB: Object Request Broker PMI data
- Server: PMI data in the whole server, you must pass recursive=true to PerfMBean
- SessionManager*: HTTP Sessions PMI data
- StatefulSessionBean*: specific EJB PMI data (BeanModule)
- StatelessSessionBean*: specific EJB PMI data (BeanModule)
- SystemMetrics: system level PMI data
- ThreadPool*: thread pool PMI data
- TransactionService: JTA Transaction PMI data
- WebModule*: Web application PMI data
- Servlet*: servlet PMI data
- WLMAppServer: Workload Management PMI data
- WebServicesService: Web services PMI data
- WSGW*: Web services gateway PMI data

To use the AdminClient API to query the MBean ObjectName for each MBean type. You can either query all the MBeans and then match the MBean type or use the query String for the type only: `String query = "WebSphere:type=mytype,node=mynode,server=myserver,*";`

Set the mytype, mynode, and myserver values accordingly. You get a Set value when you call the AdminClient class to query MBean ObjectNames. This response means that you can get multiple ObjectNames.

In the previous example, the MBean types with a star (*) mean that there can be multiple ObjectNames in a server for the same MBean type. In this case, the ObjectNames can be identified by both type and name (but mbeanIdentifier is the real UID for MBeans). However, the MBean names are not predefined. They are decided at run time based on the applications and resources. When you get multiple ObjectNames, you can construct an array of ObjectNames that you are interested in. Then you can pass the

ObjectNames to PerfMBean to get PMI data. You have the recursive and non-recursive options. The recursive option returns Stats and sub-stats objects in a tree structure while the non-recursive option returns a Stats object for that MBean only. More programming information can be found in “Developing your own monitoring applications” on page 2223.

Enterprise bean counters

Use this page as a reference for properties of enterprise bean counters.

Counters for this category report load values, response times, and life cycle activities for enterprise beans.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
CreateCount	beanModule.creates	The number of times that beans were created	3.5.5 and later	Per home	CountStatistic	Basic	Low
RemoveCount	beanModule.removes	The number of times that beans were removed	3.5.5 and later	Per home	CountStatistic	Basic	Low
PassivateCount	beanModule.passivates	The number of times that beans were passivated (entity and stateful)	3.5.5 and later	Per home	CountStatistic	Basic	Low
ActivateCount	beanModule.activates	The number of times that beans were activated (entity and stateful)	3.5.5 and later	Per home	CountStatistic	All	Low
LoadCount	beanModule.loads	The number of times that bean data was loaded from persistent storage (entity)	3.5.5 and later	Per home	CountStatistic	All	Low
StoreCount	beanModule.stores	The number of times that bean data was stored in persistent storage (entity)	3.5.5 and later	Per home	CountStatistic	All	Low
InstantiateCount	beanModule.instantiates	The number of times that bean objects were instantiated	3.5.5 and later	Per home	CountStatistic	All	Low
FreedCount	beanModule.destroys	The number of times that bean objects were freed	3.5.5 and later	Per home	CountStatistic	All	Low
Ready Count	beanModule.readyCount	The number of concurrently ready beans (entity and session). This counter was called concurrent active in Versions 3.5.5+ and 4.0.	3.5.5 and later	Per home	RangeStatistic	Basic	High
LiveCount	beanModule.concurrentLives	The number of concurrently live beans	3.5.5 and later	Per home	RangeStatistic	Extended	High

MethodResponseTime	beanModule.avgMethodRt	The average response time in milliseconds on the bean methods (home, remote, local)	3.5.5 and later	Per home	TimeStatistic	Basic	High
CreateTime	beanModule.avgCreateTime	The average time in milliseconds that a bean create call takes including the time for the load if any	5.0	Per home	TimeStatistic	All	Max
LoadTime	beanModule.loadTime	The average time in milliseconds for loading the bean data from persistent storage (entity)	5.0	Per home	TimeStatistic	All	Medium
StoreTime	beanModule.storeTime	The average time in milliseconds for storing the bean data to persistent storage (entity)	5.0	Per home	TimeStatistic	All	Medium
RemoveTime	beanModule.avgRemoveTime	The average time in milliseconds that a bean entries call takes including the time at the database, if any	5.0	Per home	TimeStatistic	All	Max
MethodCallCount	beanModule.totalMethodCalls	The total number of method calls	3.5.5 and later	Per home	CountStatistic	Basic	High
ActivationTime	beanModule.activationTime	The average time in milliseconds that a beanActivate call takes including the time at the database, if any	5.0	Per home	TimeStatistic	All	Medium
PassivationTime	beanModule.passivationTime	The average time in milliseconds that a beanPassivate call takes including the time at the database, if any	5.0	Per home	TimeStatistic	All	Medium

ActiveMethodCount	beanModule.activeMethods	The number of concurrently active methods - the number of methods called at the same time.	3.5.5 and later	Per home	TimeStatistic	All	High
RetrieveFromPoolCount	beanModule.getsFromPool	The number of calls retrieving an object from the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	All	Low
RetrieveFromPoolSuccessCount	beanModule.getsFound	The number of times that a retrieve found an object available in the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	All	Low
ReturnsToPoolCount	beanModule.returnsToPool	The number of calls returning an object to the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	Extended	Low
ReturnsDiscardCount	beanModule.returnsDiscarded	The number of times that the returning object was discarded because the pool was full (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	Extended	Low
DrainsFromPoolCount	beanModule.drainsFromPool	The number of times that the daemon found the pool was idle and attempted to clean it (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	All	Low
DrainSize	beanModule.avgDrainSize	The average number of objects discarded in each drain (entity and stateless)	3.5.5 and later	Per home and per object pool	TimeStatistic	All	Medium
PooledCount	beanModule.poolSize	The number of objects in the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	RangeStatistic	Basic	High
MessageCount	beanModule.messageCount	The number of messages delivered to the bean onMessage method (message driven beans)	5.0	Per type	CountStatistic	Basic	Low

MessageBackoutCount	beanModule.messageBackoutCount	The number of messages that failed to be delivered to the bean onMessage method (message driven beans)	5.0	Per type	CountStatistic	All	Low
WaitTime	beanModule.avgSrvSessionWaitTime	The average time to obtain a ServerSession from the pool (message-driven bean)	5.0	Per type	TimeStatistic	All	Medium
ServerSessionPoolUsage	beanModule.serverSessionUsage	The percentage of the server session pool in use (message driven)	5.0	Per type	RangeStatistic	All	High

JDBC connection pool counters

Performance Monitoring Infrastructure (PMI) collects performance data for 4.0 and 5.0 Java Database Connectivity (JDBC) data sources. For a 4.0 data source, the data source name is used. For a 5.0 data source, the Java Naming and Directory Interface (JNDI) name is used.

The JDBC connection pool counters are used to monitor the performance of JDBC data sources.

Note: With the instrumentation level set to anything other than MAX, the values may be less accurate for TimeStatistics and CountStatistics (and in the case of CountStatistics, such as numConnectionHandles, can even be negative). This is due to counters not being synchronized. Synchronizing counters is very expensive in terms of resources, so it is only done when the instrumentation level is set to MAX.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
Num creates	connectionPoolModule.numCreates	The total number of connections created	3.5.5 and later	Per connection pool	CountStatistic	All	Low
Pool size	connectionPoolModule.poolSize	The size of the connection pool	3.5.5 and later	Per connection pool	BoundedRangeStatistic	All	High
Free pool size	connectionPoolModule.freePoolSize	The number of free connections in the pool (apply to 5.0 DataSource only)	5.0	Per connection pool	BoundedRangeStatistic	Basic	High
Num allocates	connectionPoolModule.numAllocates	The total number of connections allocated	3.5.5 and later	Per connection pool	CountStatistic	All	Low
Num returns	connectionPoolModule.numReturns	The total number of connections returned	4.0 and later	Per connection pool	CountStatistic	All	Low
Concurrent waiters	connectionPoolModule.concurrentWaiters	The number of threads that are currently waiting for a connection	3.5.5 and later	Per connection pool	RangeStatistic	All	High
Faults	connectionPoolModule.faults	The total number of faults, such as timeouts, in the connection pool	3.5.5 and later	Per connection pool	CountStatistic	All	Low
Num closes	connectionPoolModule.numDestroys	The total number of connections closed.	3.5.5 and later	Per connection pool	CountStatistic	All	Low
Avg wait time (ms)	connectionPoolModule.avgWaitTime	The average waiting time in milliseconds until a connection is granted	5.0	Per connection pool	TimeStatistic	All	Medium
Avg use time (ms)	connectionPoolModule.avgUseTime	The average time a connection is used (apply to 5.0 DataSource only). Difference between the time at which the connection is allocated and returned. This value includes the JDBC operation time.	5.0	Per connection pool	TimeStatistic	All	Medium
Percent used	connectionPoolModule.percentUsed	The average percent of the pool that is in use	3.5.5 and later	Per connection pool	RangeStatistic	Basic	High
Percent maxed	connectionPoolModule.percentMaxed	The average percent of the time that all connections are in use	3.5.5 and later	Per connection pool	RangeStatistic	All	High
Prepared stmt cache discards	connectionPoolModule.prepStmtCacheDiscards	The total number of statements discarded by the least recently used (LRU) algorithm of the statement cache	4.0 and later	Per connection pool	CountStatistic	All	Low
Num managed connections	connectionPoolModule.numManagedConnections	The number of ManagedConnection objects in use for a particular connection pool (applies to V5.0 DataSource objects only)	5.0	Per connection factory	CountStatistic	All	Low
Num connection handles	connectionPoolModule.numConnectionHandles	The number of Connection objects in use for a particular connection pool (apply to 5.0 DataSource only)	5.0	Per connection factory	CountStatistic	All	Low

JDBC time	connectionPoolModule.jdbcOperationTimer	The amount of time in milliseconds spent running in the JDBC driver which includes time spent in the JDBC driver, network, and database (apply to 5.0 DataSource only)	5.0	Per data source	TimeStatistic	All	Medium

J2C connection pool counters

Use this page as a reference for properties of J2C connection pool counters.

The Java 2 Connector (J2C) connection pool counters are used to monitor J2C connection pool performance. You can find the data using the Tivoli performance viewer and clicking ***application_server*** > **J2C connection pool**.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
ManagedConnectionCount	j2cModule.numManagedConnections	The number of ManagedConnection objects in use	5.0	Per connection factory	CountStatistic	All	Low
ConnectionHandleCount	j2cModule.numConnectionHandles	The number of connections that are associated with ManagedConnections (physical connections) objects in this pool	5.0	Per connection factory	CountStatistic	All	Low
CreateCount	j2cModule.numManagedConnectionsCreated	The total number of managed connections created	5.0	Per connection factory	CountStatistic	Basic	Low
CloseCount	j2cModule.numManagedConnectionsDestroyed	The total number of managed connections destroyed	5.0	Per connection factory	CountStatistic	Basic	Low
AllocateCount	j2cModule.numManagedConnectionsAllocated	The total number of times that a managed connection is allocated to a client (the total is maintained across the pool, not per connection).	5.0	Per connection factory	CountStatistic	All	Low
FreedCount	j2cModule.numManagedConnectionsReleased	The total number of times that a managed connection is released back to the pool (the total is maintained across the pool, not per connection).	5.0	Per connection factory	CountStatistic	All	Low
FaultCount	j2cModule.faults	The number of faults, such as timeouts, in the connection pool	5.0	Per connection factory	CountStatistic	All	Low
FreePoolSize	j2cModule.freePoolSize	The number of free connections in the pool	5.0	Per connection factory	BoundedRangeStatistic	Basic	High
PoolSize	j2cModule.poolSize	Average number of managed connections in the pool.	5.0	Per connection factory	BoundedRangeStatistic	Basic	High
WaitingThreadCount	j2cModule.concurrentWaiters	Average number of threads concurrently waiting for a connection	5.0	Per connection factory	RangeStatistic	Basic	High
PercentUsed	j2cModule.percentUsed	Average percent of the pool that is in use. The value is determined by the total number of configured connections in the ConnectionPool, not the current number of connections.	5.0	Per connection factory	RangeStatistic	All	High
PercentMaxed	j2cModule.percentMaxed	Average percent of the time that all connections are in use	5.0	Per connection factory	RangeStatistic	All	High

WaitTime	j2cModule.avgWait	Average waiting time in milliseconds until a connection is granted	5.0	Per connection factory	TimeStatistic	Basic	Medium
UseTime	j2cModule.useTime	Average time in milliseconds that connections are in use	5.0	Per connection factory	TimeStatistic	Basic	Medium

Java virtual machine counters

The Java virtual machine (JVM) counters are used to monitor JVM performance. The total, used, and free heap size counters are available without any additional configuration settings. The remaining counters are available only when a Java virtual machine profiler is enabled. See Enabling the Java virtual machine profiler data for more information on this task.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
CpuUsage	jvmRuntimeModule.cpuUsage	The average percent of central processing unit (CPU) usage since the last query	6.1 and later	Per JVM	CountStatistic	Basic	Low
FreeMemory	jvmRuntimeModule.freeMemory	The free memory in the JVM run time	3.5.5 and later	Per JVM	CountStatistic	Extended	Low
UsedMemory	jvmRuntimeModule.usedMemory	The used memory in the JVM run time	3.5.5 and later	Per JVM	CountStatistic	Basic	Low
HeapSize	jvmRuntimeModule.totalMemory	The total memory in the JVM run time	3.5.5 and later	Per JVM	BoundedRangeStatistic. The upperBound and lowerBound are not implemented for the Total memory counter.	Basic	High
UpTime	jvmRuntimeModule.upTime	The amount of time that the JVM is running	5.0	Per JVM	CountStatistic	Basic	Low
GCCount	jvmRuntimeModule.numGcCalls	The number of garbage collection calls. This counter is not available unless the JVM profiler is enabled. See Enabling the JVM profiler data.	4.0 and later	Per JVM	CountStatistic	All	Max
GCIntervalTime	jvmRuntimeModule.avgTimeBetweenGcCalls	The average garbage collection value in seconds between two garbage collections. This counter is not available unless the JVM profiler is enabled. See Enabling the JVM profiler data.	4.0 and later	Per JVM	TimeStatistic	All	Max
GCTime	jvmRuntimeModule.avgGcDuration	The average duration of a garbage collection. This counter is not available unless the JVM profiler is enabled. See Enabling the JVM profiler data.	4.0 and later	Per JVM	TimeStatistic	All	Max
WaitsForLockCount	jvmRuntimeModule.numWaitsForLock	The number of times that a thread waits for a lock. This counter is not available unless the JVM profiler is enabled. See Enabling the JVM profiler data.	4.0 and later	Per JVM	CountStatistic	All	Max
WaitForLockTime	jvmRuntimeModule.avgTimeWaitForLock	The average time that a thread waits for a lock. This counter is not available unless the JVM profiler is enabled. See Enabling the JVM profiler data.	4.0 and later	Per JVM	TimeStatistic	All	Max

ThreadStartedCount	jvmRuntimeModule.numThreadsStarted	The number of threads started. This counter is not available unless the JVM profiler is enabled. See Enabling the JVM profiler data.	4.0 and later	Per JVM	CountStatistic	All	Max
ThreadEndedCount	jvmRuntimeModule.numThreadsDead	The number of failed threads. This counter is not available unless the JVM profiler is enabled. See Enabling the JVM profiler data.	4.0 and later	Per JVM	CountStatistic	All	Max

ObjectAllocateCount	jvmRuntimeModule.numObjectsAllocated	The number of objects that are allocated in the heap. This counter is not available unless the -XrunpmiJvmpiProfiler option is set when starting the JVM.	From 4.0 to 6.0.x. It is deprecated in 6.1.	Per JVM	CountStatistic	All	Max
ObjectMovedCount	jvmRuntimeModule.numObjectsMoved	The number of objects in the heap. This counter is not available unless the -XrunpmiJvmpiProfiler option is set when starting the JVM.	From 4.0 to 6.0.x. It is deprecated in 6.1.	Per JVM	CountStatistic	All	Max
ObjectFreedCount	jvmRuntimeModule.numObjectsFreed	The number of objects freed in the heap. This counter is not available unless the -XrunpmiJvmpiProfiler option is set when starting the JVM.	From 4.0 to 6.0.x. It is deprecated in 6.1.	Per JVM	CountStatistic	All	Max

Important: The statistics that are gathered through the JVM Tool Interface (JVMTI) is different between the JVM provided by IBM and the Sun HotSpot-based JVM, including Sun HotSpot JVM on Solaris and HP's JVM for HP-UX.

Object Request Broker counters

Use this page as a reference for properties of Object Request Broker counters.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
LookupTime	orbPerfModule.referenceLookupTime	The time (in milliseconds) to look up an object reference before method dispatch can be carried out. An excessively long time can indicate an EJB container lookup problem.	5.0	Object Request Broker (ORB)	TimeStatistic	All	Medium
RequestCount	orbPerfModule.totalRequests	The total number of requests sent to the ORB	5.0	ORB	CountStatistic	All	Low
ConcurrentRequestCount	orbPerfModule.concurrentRequests	The number of requests that are concurrently processed by the ORB	5.0	ORB	RangeStatistic	All	High
ProcessingTime	orbPerfModule.interceptors.processingTime	The time (in milliseconds) it takes a registered portable interceptor to run	5.0	Per interceptor	TimeStatistic	All	Medium

Servlet session counters

Use this page as a reference for properties of servlet session counters.

Data counters for this category contain usage information for HTTP sessions.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
CreateCount	servletSessionsModule.createdSessions	The number of sessions that were created	3.5.5 and later	Per Web application	CountStatistic	All	Low
InvalidateCount	servletSessionsModule.invalidatedSessions	The number of sessions that were invalidated	3.5.5 and later	Per Web application	CountStatistic	All	Low
LifeTime	servletSessionsModule.sessionLifeTime	The average session life time in milliseconds (time invalidated - time created)	3.5.5 and later	Per Web application	TimeStatistic	Extended	Medium
ActiveCount	servletSessionsModule.activeSessions	The number of concurrently active sessions. A session is active if the WebSphere Application Server is currently processing a request that uses that session.	3.5.5 and later	Per Web application	RangeStatistic	All	High
LiveCount	servletSessionsModule.liveSessions	The number of sessions that are currently cached in memory	5.0 and later	Per Web application	RangeStatistic	Basic	High
NoRoomForNewSessionCount	servletSessionsModule.noRoomForNewSession	Applies only to session in memory with AllowOverflow=false. The number of times that a request for a new session cannot be handled because it exceeds the maximum session count.	5.0	Per Web application	CountStatistic	Extended	Low
CacheDiscardCount	servletSessionsModule.cacheDiscards	The number of session objects that have been forced out of the cache. A least recently used (LRU) algorithm removes old entries to make room for new sessions and cache misses. Applicable only for persistent sessions.	5.0	Per Web application	CountStatistic	All	Low
ExternalReadTime	servletSessionsModule.externalReadTime	The time (milliseconds) taken in reading the session data from the persistent store. For multirow sessions, the metrics are for the attribute; for single row sessions, the metrics are for the entire session. Applicable only for persistent sessions. When using a JMS persistent store, you can choose to serialize the replicated data. If you choose not to serialize the data, the counter is not available.	5.0	Per Web application	TimeStatistic	Extended	Medium

ExternalReadSize	servletSessionsModule.externalReadSize	Size of the session data read from persistent store. Applicable only for (serialized) persistent sessions; similar to external Read Time.	5.0	Per Web application	TimeStatistic	Extended	Medium
ExternalWriteTime	servletSessionsModule.externalWriteTime	The time (milliseconds) taken to write the session data from the persistent store. Applicable only for (serialized) persistent sessions. Similar to external Read Time.	5.0	Per Web application	TimeStatistic	Extended	Medium
ExternalWriteSize	servletSessionsModule.externalWriteSize	The size of the session data written to persistent store. Applicable only for (serialized) persistent sessions. Similar to external Read Time.	5.0	Per Web application	TimeStatistic	Extended	Medium
AffinityBreakCount	servletSessionsModule.affinityBreaks	The number of requests that are received for sessions that were last accessed from another Web application. This value can indicate failover processing or a corrupt plug-in configuration.	5.0	Per Web application	CountStatistic	All	Low
SessionObjectSize	servletSessionsModule.serializableSessObjSize	The size in bytes of (the serializable attributes of) in-memory sessions. Only session objects that contain at least one serializable attribute object is counted. A session can contain some attributes that are serializable and some that are not. The size in bytes is at a session level.	5.0	Per Web application	TimeStatistic	All	Max
TimeSinceLastActivated	servletSessionsModule.timeSinceLastActivated	The time difference in milliseconds between previous and current access time stamps. Does not include session time out.	5.0	Per Web application	TimeStatistic	All	Medium
TimeoutInvalidationCount	servletSessionsModule.invalidatedViaTimeout	The number of sessions that are invalidated by timeout.	5.0	Per Web application	CountStatistic	All	Low
ActivateNonExistSessionCount	servletSessionsModule.activateNonExistSessions	The number of requests for a session that no longer exists, presumably because the session timed out. Use this counter to help determine if the timeout is too short.	5.0	Per Web application	CountStatistic	All	Low

Transaction counters

Use this page as a reference for properties of transaction counters.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
GlobalBegunCount	transactionModule.globalTransBegun	The total number of global transactions started on the server	4.0 and later	Per transaction manager or server	CountStatistic	Extended	Low
GlobalInvolvedCount	transactionModule.globalTransInvolved	The total number of global transactions involved on the server (for example, begun and imported)	4.0 and later	Per transaction manager or server	CountStatistic	All	Low
LocalBegunCount	transactionModule.localTransBegun	The total number of local transactions started on the server	4.0 and later	Per transaction manager or server	CountStatistic	Extended	Low
ActiveCount	transactionModule.activeGlobalTrans	The number of concurrently active global transactions	3.5.5 and later	Per transaction manager or server	CountStatistic	Basic	Low
LocalActiveCount	transactionModule.activeLocalTrans	The number of concurrently active local transactions	4.0 and later	Per transaction manager or server	CountStatistic	All	Low
GlobalTranTime	transactionModule.globalTranDuration	The average duration of global transactions	3.5.5 and later	Per transaction manager or server	TimeStatistic	Extended	Medium
LocalTranTime	transactionModule.localTranDuration	The average duration of local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Extended	Medium
GlobalBeforeCompletionTime	transactionModule.globalBeforeCompletionDuration	The average duration of before_completion for global transactions	4.0 and later	per transaction manager or server	TimeStatistic	All	Medium
GlobalCommitTime	transactionModule.globalCommitDuration	The average duration of commit for global transactions	4.0 and later	Per transaction manager or server	TimeStatistic	All	Medium
GlobalPrepareTime	transactionModule.globalPrepareDuration	The average duration of prepare for global transactions	4.0 and later	Per transaction manager or server	TimeStatistic	All	Medium
LocalBeforeCompletionTime	transactionModule.localBeforeCompletionDuration	The average duration of before_completion for local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	All	Medium
LocalCommitTime	transactionModule.localCommitDuration	The average duration of commit for local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	All	Medium
CommittedCount	transactionModule.globalTransCommitted	The total number of global transactions committed	3.5.5 and later	Per transaction manager or server	CountStatistic	Basic	Low
RolledbackCount	transactionModule.globalTransRolledBack	The total number of global transactions rolled back	3.5.5 and later	Per transaction manager or server	CountStatistic	Basic	Low
OptimizationCount	transactionModule.numOptimization	The number of global transactions converted to single phase for optimization	4.0 and later	Per transaction manager or server	CountStatistic	All	Low
LocalCommittedCount	transactionModule.localTransCommitted	The number of local transactions committed	4.0 and later	Per transaction manager or server	CountStatistic	All	Low

LocalRolledbackCount	transactionModule.localTransRolledBack	The number of local transactions rolled back	4.0 and later	Per transaction manager or server	CountStatistic	All	Low
GlobalTimeoutCount	transactionModule.globalTransTimeout	The number of global transactions timed out	4.0 and later	Per transaction manager or server	CountStatistic	Extended	Low
LocalTimeoutCount	transactionModule.localTransTimeout	The number of local transactions timed out	4.0 and later	Per transaction manager or server	CountStatistic	Extended	Low

Thread pool counters

Use this page as a reference for properties of thread pool counters.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
CreateCount	threadPoolModule.threadCreates	The total number of threads created	3.5.5 and later	Per thread pool	CountStatistic	All	Low
DestroyCount	threadPoolModule.threadDestroys	The total number of threads destroyed	3.5.5 and later	Per thread pool	CountStatistic	All	Low
ActiveCount	threadPoolModule.activeThreads	The number of concurrently active threads	3.5.5 and later	Per thread pool	RangeStatistic	Extended	High
PoolSize	threadPoolModule.poolSize	The average number of threads in pool	3.5.5 and later	Per thread pool	BoundedRangeStatistic	Basic	High
PercentMaxed	threadPoolModule.percentMaxed	The average percent of the time that all threads are in use	3.5.5 and later	Per thread pool	RangeStatistic	All	High
DeclaredthreadHungCount	threadPoolModule.declaredThreadHung	The number of threads declared hung	5.1.1 and later	Per thread pool	CountStatistic	All	Max
ClearedThreadHangCount	threadPoolModule.declaredThreadHangCleared	The number of thread hangs cleared	5.1.1 and later	Per thread pool	CountStatistic	All	Max
ConcurrentHungThreadCount	threadPoolModule.concurrentlyHungThreads	The number of concurrently hung threads	5.1.1 and later	Per thread pool	RangeStatistic	All	Max
ActiveTime	threadPoolModule.activeTime	The average time in milliseconds the threads are in active state	5.1.1 and later	Per thread pool	TimeStatistic	All	Max

Web application counters

Data counters for this category contain information for the selected server.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
LoadedServletCount	webAppModule.numLoadedServlets	The number of loaded servlets	3.5.5 and later	Per Web application	CountStatistic	All	Low
ReloadCount	webAppModule.numReloads	The number of reloaded servlets	3.5.5 and later	Per Web application	CountStatistic	All	Low
RequestCount	webAppModule.servlets.totalRequests	Total number of requests that a servlet processed	3.5.5 and later	Per servlet	CountStatistic	Basic	Low
ConcurrentRequests	webAppModule.servlets.concurrentRequests	The number of requests that are concurrently processed	3.5.5 and later	Per servlet	RangeStatistic	Extended	High
ServiceTime	webAppModule.servlets.responseTime	The response time, in milliseconds, of a servlet request	3.5.5 and later	Per servlet	TimeStatistic	Basic	Medium
ErrorCount	webAppModule.servlets.numErrors	Total number of errors in a servlet or JavaServer Page (JSP)	3.5.5 and later	Per servlet	CountStatistic	Extended	Low

Workload Management counters

Use this page as a reference for properties of workload management counters.

Data counters for this category contain information for workload management.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
Num IIOP requests	wlmModule.server.numIncomingRequests	The total number of incoming IIOP requests to an application server	5.0	Per server	CountStatistic	Extended	Low
Num strong affinity IIOP requests	wlmModule.server.numIncomingStrongAffinityRequests	The number of incoming IIOP requests to an application server that are based on a strong affinity. A strong affinity request is defined as a request that must be serviced by this application server because of a dependency that resides on the server. This request could not successfully be serviced on another member in the server cluster.	5.0	Per server	CountStatistic	All	Low
Num no affinity IIOP requests	wlmModule.server.numIncomingNonAffinityRequests	The number of incoming IIOP requests to an application server based on no affinity. This request was sent to this server based on workload management selection policies that were decided in the Workload Management (WLM) run time of the client.	5.0	Per server	CountStatistic	All	Low
Num non-WLM enabled IIOP requests	wlmModule.server.numIncomingNonWLMObjectRequests	The number of incoming IIOP requests to an application server that came from a client that does not have the WLM run time present or where the object reference on the client was flagged not to participate in workload management.	5.0	Per server	CountStatistic	All	Low
Num server cluster updates	wlmModule.server.numServerClusterUpdates	The number of times initial or updated server cluster data is sent to a server member from the deployment manager. This metric determines how often cluster information is being propagated.	5.0	Per server	CountStatistic	All	Low
Num of WLM clients serviced	wlmModule.server.numOfWLMClientsServiced	The number of WLM enabled clients that have been serviced by this application server.	5.0	Per server	CountStatistic	All	Low
Num concurrent requests	wlmModule.server.numOfConcurrentRequests	The number of remote IIOP requests currently being processed by this server	5.0	Per server	RangeStatistic	Extended	High
Server response time	wlmModule.server.serverResponseTime	The response time (in milliseconds) of IIOP requests being serviced by an application server. The response time is calculated based on the time the request is received to the time when the reply is sent back out.	5.0	Per server	TimeStatistic	Extended	Medium

Num outgoing IIOP requests	wlmModule.client.numOfOutgoingRequests	The total number of outgoing IIOP requests being sent from a client to an application server	5.0	Per WLM	CountStatistic	All	Low
Num server cluster updates	wlmModule.client.numClientClusterUpdates	The number of times initial or updated server cluster data is sent to a WLM enabled client from server cluster member. Use this metric to determine how often cluster information is being propagated.	5.0	Per WLM	CountStatistic	All	Low
Client response time	wlmModule.client.clientResponseTime	The response time (in milliseconds) of IIOP requests being sent from a client. The response time is calculated based on the time the request is sent from the client to the time the reply is received from the server.	5.0	Per WLM	TimeStatistic	All	Medium

System counters

Use this page as a reference for properties of system counters.

Counters for this category contain information for a machine (node).

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
CPUUsageSinceLastMeasurement	systemModule.cpuUtilization	The average system CPU utilization taken over the time interval since the last reading. Because the first call is required to perform initialization, a value such as 0, which is not valid, will be returned. All subsequent calls return the expected value. On SMP machines, the value returned is the utilization averaged over all CPUs.	5.0	Per node	CountStatistic	Basic	Low
FreeMemory	systemModule.freeMemory	The amount of real free memory available on the system. Real memory that is not allocated is only a lower bound on available real memory, since many operating systems take some of the otherwise unallocated memory and use it for additional I/O buffering. The exact amount of buffer memory which can be freed up is dependent on both the platform and the application(s) running on it.	5.0	Per node	CountStatistic	All	Low
CPUUsageSinceServerStarted	systemModule.avgCpuUtilization	The average percent CPU Usage that is busy after the server is started.	5.0	Per node	TimeStatistic	Extended	Medium

Dynamic cache counters

Use this page as a reference for properties of dynamic cache counters.

You can use the Performance Monitoring Infrastructure (PMI) data for Dynamic Cache to monitor the behavior and performance of the dynamic cache service. For information on the functions and usages of dynamic cache, refer to “Task overview: Using the dynamic cache service to improve performance” on page 1913.

Use the DynaCache MBean to access the related data and display it under Dynamic Cache in TPV.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
MaxInMemoryCacheEntryCount	cacheModule..maxInMemoryCacheEntryCount	The maximum number of in-memory cache entries. It is located under servlet instance or object instance.	5.0 and later	Per Server	CountStatistic	All	Low
InMemoryCacheEntryCount	cacheModule.inMemoryCacheEntryCount	The current number of in-memory cache entries. It is located under servlet instance and object instance.	5.0 and later	Per Server	CountStatistic	All	Low
HitsInMemoryCount	cacheModule.hitsInMemoryCount	The number of requests for cacheable objects that are served from memory. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
HitsOnDiskCount	cacheModule.hitsOnDiskCount	The number of requests for cacheable objects that are served from disk. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
ExplicitInvalidationCount	cacheModule.explicitInvalidationCount	The number of explicit invalidations. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
LruInvalidationCount	cacheModule.lruInvalidationCount	The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low

TimeoutInvalidationCount	cacheModule.timeoutInvalidationCount	The number of cache entries that are removed from memory and disk because their timeout has expired. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
InMemoryAndDiskCacheEntryCount	cacheModule.inMemoryAndDiskCacheEntryCount	The current number of used cache entries in memory and disk. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
RemoteHitCount	cacheModule.remoteHitCount	The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
MissCount	cacheModule.missCount	The number of requests for cacheable objects that were not found in the cache. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
ClientRequestCount	cacheModule.clientRequestCount	The number of requests for cacheable objects that are generated by applications running on this application server. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low

DistributedRequestCount	cacheModule.distributedRequestCount	The number of requests for cacheable objects that are generated by cooperating caches in this replication domain. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
ExplicitMemoryInvalidationCount	cacheModule.explicitMemoryInvalidationCount	The number of explicit invalidations resulting in the removal of an entry from memory. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
ExplicitDiskInvalidationCount	cacheModule.explicitDiskInvalidationCount	The number of explicit invalidations resulting in the removal of an entry from disk. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
LocalExplicitInvalidationCount	cacheModule.localExplicitInvalidationCount	The number of explicit invalidations generated locally, either programmatically or by a cache policy. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
RemoteExplicitInvalidationCount	cacheModule.remoteExplicitInvalidationCount	The number of explicit invalidations received from a cooperating Java virtual machine in this replication domain. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low

RemoteCreationCount	cacheModule.remoteCreationCount	The number of cache entries that are received from cooperating dynamic caches. For servlet instance, it is located under template group. For object instance, it is located under object group.	5.0 and later	Per Server	CountStatistic	All	Low
ObjectsOnDisk	cacheModule.objectsOnDisk	The current number of cache entries on disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
HitsOnDisk	cacheModule.hitsOnDisk	The number of requests for cacheable objects that are served from disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
ExplicitInvalidationsFromDisk	cacheModule.explicitInvalidationFromDisk	The number of explicit invalidations resulting in the removal of entries from disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
TimeoutInvalidationsFromDisk	cacheModule.timeoutInvalidationsFromDisk	The number of disk timeouts. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
PendingRemovalFromDisk	cacheModule.pendingRemovalFromDisk	The current number of pending entries that are to be removed from disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
DependencyIDsOnDisk	cacheModule.dependencyIdsOnDisk	The current number of dependency ID that are on disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
DependencyIDsBufferedForDisk	cacheModule.dependencyIdsBufferedForDisk	The current number of dependency IDs that are buffered for the disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
DependencyIDsOffloadedToDisk	cacheModule.dependencyIdsOffloadedToDisk	The number of dependency IDs that are offloaded to disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low

DependencyIDBasedInvalidationsFromDisk	cacheModule.dependencyIdBasedInvalidationsFromDisk	The number of dependency ID-based invalidations. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
TemplatesOnDisk	cacheModule.templatesOnDisk	The current number of templates that are on disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
TemplatesBufferedForDisk	cacheModule.templatesBufferedForDisk	The current number of templates that are buffered for the disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
TemplatesOffloadedToDisk	cacheModule.templatesOffloadedToDisk	The number of templates that are offloaded to disk. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
TemplateBasedInvalidationsFromDisk	cacheModule.templateBasedInvalidationsFromDisk	The number of template-based invalidations. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
GarbageCollectorInvalidationsFromDisk	cacheModule.garbageCollectorInvalidationsFromDisk	The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low
OverflowInvalidationsFromDisk	cacheModule.overflowInvalidationsFromDisk	The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit. It is located under disk group.	6.1 and later	Per Server	CountStatistic	All	Low

Web services gateway counters

Use this page as a reference for properties of Web services gateway counters.

Counters for this category contain information for Web services gateways (WSGW). Examples include the number of synchronous and asynchronous requests and responses.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
SynchronousRequestCount	wsgwModule.synchronousRequests	The number of synchronous requests made.	5.0	Per Web service	CountStatistic	All	Low
SynchronousResponseCount	wsgwModule.synchronousResponses	The number of synchronous responses made.	5.0	Per Web service	CountStatistic	All	Low
AsynchronousRequestCount	wsgwModule.asynchronousRequests	The number of asynchronous requests made.	5.0	Per Web service	CountStatistic	All	Low
AsynchronousResponseCount	wsgwModule.asynchronousResponses	The number of asynchronous responses made.	5.0	Per Web service	CountStatistic	All	Low

Web services counters

Use this page as a reference for properties of Web services counters.

Counters for this category contain information for the Web services.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
LoadedWebServiceCount	webServicesModule.numLoadedServices	The number of loaded Web services	5.02 and above	Per service	CountStatistic	All	Low
ReceivedRequestCount	webServicesModule.services.numberReceived	The number of requests the service received	5.02 and above	Per Web service	CountStatistic	All	Low
DispatchedRequestCount	webServicesModule.services.numberDispatched	The number of requests the service dispatched or delivered	5.02 and above	Per Web service	CountStatistic	All	Low
ProcessedRequestCount	webServicesModule.services.numberSuccessfull	The number of requests the service successfully processed	5.02 and above	Per Web service	TimeStatistic	All	Low
ResponseTime	webServicesModule.services.responseTime	The average response time, in milliseconds, for a successful request	5.02 and above	Per Web service	TimeStatistic	All	High
RequestResponseTime	webServicesModule.services.requestResponseTime	The average response time, in milliseconds, to prepare a request for dispatch	5.02 and above	Per Web service	TimeStatistic	All	Medium
DispatchResponseTime	webServicesModule.services.dispatchResponseTime	The average response time, in milliseconds, to dispatch a request	5.02 and above	Per Web service	TimeStatistic	All	Medium
ReplyResponseTime	webServicesModule.services.replyResponseTime	The average response time, in milliseconds, to prepare a reply after dispatch	5.02 and above	Per Web service	TimeStatistic	All	Medium
PayloadSize	webServicesModule.services.size	The average payload size in bytes of a received request or reply	5.02 and above	Per Web service	TimeStatistic	All	Medium
RequestPayloadSize	webServicesModule.services.requestSize	The average payload size in bytes of a request	5.02 and above	Per Web service	TimeStatistic	All	Medium
ReplyPayloadSize	webServicesModule.services.replySize	The average payload size in bytes of a reply	5.02 and above	Per Web service	TimeStatistic	All	Medium

Alarm Manager counters

Use this page as a reference for properties of alarm manager counters.

Counters for this category contain information for the Alarm Manager.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
AlarmsCreatedCount	alarmManagerModule.numCreates.name	The total number of alarms created by all asynchronous scopes for this WorkManager	5.0 and later	Per WorkManager	CountStatistic	All	High
AlarmsCancelledCount	alarmManagerModule.numCancelled.name	The number of alarms cancelled by the application	5.0 and later	Per WorkManager	CountStatistic	All	High
AlarmsFiredCount	alarmManagerModule.numFired.name	The number of alarms fired	5.0 and later	Per WorkManager	CountStatistic	All	High
AlarmLatencyDuration	alarmManagerModule.latency.name	The latency of alarms fired in milliseconds	5.0 and later	Per WorkManager	RangeStatistic	All	High
AlarmsPendingSize	alarmManagerModule.alarmsPending.name	The number of alarms waiting to fire	5.0 and later	Per WorkManager	RangeStatistic	All	High
AlarmRate	alarmManagerModule.alarmsPerSecond.name	The number of alarms firing per second	5.0 and later	Per WorkManager	RangeStatistic	All	High

Object Pool counters

Use this page as a reference for properties of Object Pool counters.

Counters for this category contain information for Object Pools.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
ObjectsCreatedCount	objectPoolModule.numCreates.name	The total number of objects created	5.0 and later	Per ObjectPool	CountStatistic	All	High
ObjectsAllocatedCount	objectPoolModule.numAllocates.name	The number of objects requested from the pool	5.0 and later	Per ObjectPool	CountStatistic	All	High
ObjectsReturnedCount	objectPoolModule.numReturns.name	The number of objects returned to the pool	5.0 and later	Per ObjectPool	CountStatistic	All	High
IdleObjectsSize	objectPoolModule.poolSize.name	The average number of idle object instances in the pool	5.0 and later	Per ObjectPool	RangeStatistic	All	High

Scheduler counters

Use this page as a reference for properties of Scheduler counters.

Counters for this category contain information for the Scheduler service.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
TaskFailureCount	schedulerModule.failedTasks.name	The number of tasks that failed to run	5.0 and later	Per Scheduler	CountStatistic	All	High
TaskFinishCount	schedulerModule.executedTasks.name	The number of tasks ran successfully	5.0 and later	Per Scheduler	CountStatistic	All	High
PollCount	schedulerModule.totalPolls.name	The number of poll cycles completed for all daemon threads	5.0 and later	Per Scheduler	CountStatistic	All	High
TaskFinishRate	schedulerModule.tasksPerSec.name	The number of tasks run per second	5.0 and later	Per Scheduler	RangeStatistic	All	High
TaskCollisionRate	schedulerModule.collisionsPerSec.name	The number of collisions encountered per second between competing poll daemons	5.0 and later	Per Scheduler	RangeStatistic	All	High
PollQueryDuration	schedulerModule.queryTime.name	The start time in milliseconds for each poll daemon thread's database poll query	5.0 and later	Per Scheduler	RangeStatistic	All	High
RunDuration	schedulerModule.execTime.name	The time in milliseconds taken to run a task.	5.0 and later	Per Scheduler	RangeStatistic	All	High
TaskExpirationRate	schedulerModule.taskLoadPerPoll.name	The number of tasks in a poll query	5.0 and later	Per Scheduler	RangeStatistic	All	High
TaskDelayDuration	schedulerModule.execLatency.name	The period of time in seconds that the task is delayed	5.0 and later	Per Scheduler	RangeStatistic	All	High
PollDuration	schedulerModule.pollTime.name	The number of seconds between poll cycles	5.0 and later	Per Scheduler	RangeStatistic	All	High
TaskRunRate	schedulerModule.taskExecPerPoll.name	The number of tasks run by each poll daemon thread. (Multiply this by the number of poll daemon threads to get the tasks run per effective poll cycle.)	5.0 and later	Per Scheduler	RangeStatistic	All	High

High availability manager counters

PMI maintains statistical data within the entire WebSphere Application Server domain, including multiple nodes and servers.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
Number of local groups	hamanagermodule.numLocalGroups	The total number of local groups.	6.0 and above	Per server	RangeStatistic	All	High
Group state rebuild time	hamanagermodule.rebuildTime	Time taken in milliseconds to rebuild the global group state. During the rebuild time, no fail-over can happen. If this time is too high and is unacceptable for the desired availability, you may want to increase the number of coordinators. For proper operation of this counter, you must host the active coordinator in an application server other than the deployment manager.	6.0 and above	Per server	TimeStatistic	All	High
Number of bulletin-board subjects	hamanagermodule.bbMgrNumSubjects	The total number of subjects managed.	6.0 and above	Per server	RangeStatistic	All	High
Number of bulletin-board subscriptions	hamanagermodule.bbMgrNumSubscriptions	The total number of bulletin-board subscriptions.	6.0 and above	Per server	RangeStatistic	All	High
Bulletin-board rebuild time	hamanagermodule.bbMgrRebuildTime	Time taken in milliseconds to rebuild the global state of the bulletin-board. During this time no messages will be received by the subscribers. If this time is too high, and is unacceptable, you may want to increase the number of coordinators. For proper operation of this counter, you must host the active coordinator in an application server other than the deployment manager.	6.0 and above	Per server	TimeStatistic	All	High
Number of local bulletin-board subjects	hamanagermodule.bbLocalNumSubjects	The total number of subjects being posted to locally. The number includes the proxy postings (if any) done by the core group bridge service on behalf of servers belonging to different WebSphere cells.	6.0 and above	Per server	RangeStatistic	All	High

DCS stack counters

PMI maintains statistical data within the entire WebSphere Application Server domain, including multiple nodes and servers.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead
Number of message buffer reallocations	DCSStats.numOfReallocs	Number of message buffer reallocations due to inadequate buffer size. If this number is larger than 20 percent of the number of sent messages, you may want to contact IBM Support	6.0 and above	Per DCS stack	CountStatistic	All	Medium
Outgoing message size	DCSStats.outgoingMessageSize	Minimal, maximal, and average size (in bytes) of the messages that were sent through the DCS stack	6.0 and above	Per DCS stack	AverageStatistic	All	High
Number of sent messages	DCSStats.outgoingMessageCounter	Number of messages sent through the DCS stack	6.0 and above	Per DCS stack	CountStatistic	All	High
Incoming message size	DCSStats.incomingMessageSize	Minimal, maximal and average size (in bytes) of the messages that were received by the DCS stack	6.0 and above	Per DCS stack	AverageStatistic	All	High
Number of received messages	DCSStats.incomingMessageCounter	Number of messages received by the DCS stack	6.0 and above	Per DCS stack	CountStatistic	All	High
Amount of time needed for the synchronization procedure to complete	DCSStats.vsCompleteCurrentTime	Amount of time needed to guarantee that all view members are synchronized.	6.0 and above	Per DCS stack	TimeStatistic	All	High
Number of times that the synchronization procedure timed out	DCSStats.vsTimetoutExpiredCounter	Number of times that the synchronization procedure timed out.	6.0 and above	Per DCS stack	CountStatistic	All	Medium
Number of times that a high severity congestion event for outgoing messages was raised	DCSStats.transmitterCongestedCounter	Number of times that a high severity congestion event for outgoing messages was raised.	6.0 and above	Per DCS stack	CountStatistic	All	Medium
Coalesce Time	DCSStats.coalesceTime	Measures the amount of time it actually takes to coalesce a view.	6.0 and above	Per DCS stack	TimeStatistic	All	Medium
Join View Change Time	DCSStats.mergeTime	Measures the time to do a merge view change. The DCS stack is blocked during this time.	6.0 and above	Per DCS stack	TimeStatistic	All	High
Remove View Change Time	DCSStats.splitTime	Measures the time to do a split view change.The DCS stack is blocked during this time.	6.0 and above	Per DCS stack	TimeStatistic	All	High
Number of suspicions	DCSStats.suspectCounter	Measures the number of times that the local member suspected other members.	6.0 and above	Per DCS stack	CountStatistic	All	High
Number of view changes	DCSStats.viewCounter	Number of times that this member underwent view changes.	6.0 and above	Per DCS stack	CountStatistic	All	Medium
View group size	DCSStats.groupSize	Measures the size of the group the local member belongs to.	6.0 and above	Per DCS stack	AverageStatistic	All1	Medium

PortletContainer PMI counters

Use this page as a reference for properties of PortletContainer PMI counters.

Counter definitions

These counters are the PMI measurement points for a portlet and a portlet application.

Name	Key	Description	Version	Unit	Type	Level	Overhead
totalRequests	portletModule.totalRequests	The number of requests made to the portlet module.	6.1	unit.none	CountStatistic	All	Low
numErrors	portletModule.numErrors	The number of errors reported from the portlet module.	6.1	unit.none	CountStatistic	All	Low
concurrentRequests	portletModule.concurrentRequests	The number of concurrent requests.	6.1	unit.none	CountStatistic	All	High
renderResponseTime	portletModeule.renderResponseTime	The render response time.	6.1	unit.ms	CountStatistic	All	Medium
actionResponseTime	portletModule.actionResponseTime	The action response time.	6.1	unit.ms	CountStatistic	All	Medium
numLoadedPortlets	portletAppModule.numLoadedPortlets	The number of portlets loaded.	6.1	unit.none	CountStatistic	All	Low

Related tasks

"Task overview: Managing portlets" on page 175
You can use this task to manage deployed portlet applications.

Related reference

"PMI data organization" on page 2108
Use this page as a general overview of monitoring, data collection, and counters using Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV).

Service Integration Bus (SIB) and Messaging counters

These counters are part of Performance Monitoring Infrastructure (PMI), which provides server-side monitoring and a client-side API to retrieve performance data.

For information on SIB counters, see the following articles:

- "MessageStore Statistics"
- "Mediation Framework Statistics" on page 2175
- "Message Processor Statistics" on page 2178
- "Communications statistics" on page 2188

MessageStore Statistics:

Consult this information for properties of MessageStore statistics.

Counter definitions*:*

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache

Name	Key	Description	Version	Granularity	Type	Level
CacheAddStoredCount	MessageStoreStats.CacheAddStoredCount	The number of items that have been added to the message store during the current session that are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheAddNotStoredCount	MessageStoreStats.CacheAddNotStoredCount	The number of items that have been added to the message store during the current session that are not persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheUpdateStoredCount	MessageStoreStats.CacheUpdateStoredCount	The number of items that have been updated in the message store during the current session that are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheUpdateNotStoredCount	MessageStoreStats.CacheUpdateNotStoredCount	The number of items that have been updated in the message store during the current session that not persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheRemoveStoredCount	MessageStoreStats.CacheRemoveStoredCount	The number of items that have been removed from the message store during the current session that are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheRemoveNotStoredCount	MessageStoreStats.CacheRemoveNotStoredCount	The number of items that have been removed from the message store during the current session that are not persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheRestoreCount	MessageStoreStats.CacheRestoreCount	The number of items restored to memory from persistence during the current session	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheCurrentStoredCount	MessageStoreStats.CacheCurrentStoredCount	The number of items currently in the dynamic memory cache which are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheCurrentNotStoredCount	MessageStoreStats.CacheCurrentNotStoredCount	The number of items currently in the dynamic memory cache which are never persisted	6.0 and later	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache

CacheCurrentStoredByteCount	MessageStoreStats.CacheCurrentStoredByteCount	The total of the declared sizes of all items currently in the dynamic memory cache which are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheCurrentNotStoredByteCount	MessageStoreStats.CacheCurrentNotStoredByteCount	The current total of the declared sizes of all items in the dynamic memory cache which are never persisted	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheTotalStoredCount	MessageStoreStats.CacheTotalStoredCount	The total number of items which have been added to the dynamic memory cache during the current session which are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheTotalNotStoredCount	MessageStoreStats.CacheTotalNotStoredCount	The total number of items which have been added to the dynamic memory cache during the current session which are never persisted in cache	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheTotalStoredByteCount	MessageStoreStats.CacheTotalStoredByteCount	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheTotalNotStoredByteCount	MessageStoreStats.CacheTotalNotStoredByteCount	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are never persisted in cache	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheStoredDiscardCount	MessageStoreStats.CacheStoredDiscardCount	The total number of items which have been discarded from the dynamic memory cache during the current session which are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheNotStoredDiscardCount	MessageStoreStats.CacheNotStoredDiscardCount	The total number of items which have been discarded from the dynamic memory cache during the current session which are never persisted	6.0 and later	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache

CacheStoredDiscardByteCount	MessageStoreStats.CacheStoredDiscardByteCount	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheNotStoredDiscardByteCount	MessageStoreStats.CacheNotStoredDiscardByteCount	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are never persisted	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheStoredRefusalCount	MessageStoreStats.CacheStoredRefusalCount	The total number of items which have been refused entry to the dynamic memory cache during the current session which are either persistent or potentially persistent	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheNotStoredRefusalCount	MessageStoreStats.CacheNotStoredRefusalCount	The total number of items which have been refused entry to the dynamic memory cache during the current session which are never persisted	6.0 and later	Per Messaging Engine	CountStatistic	High
CacheStreamSpillingCount	MessageStoreStats.CacheStreamSpillingCount	Number of streams currently spilling potentially persistent items	6.0 and later	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache

Name	Key	Description	Version	Granularity	Type	Level
SpillDispatcherRequestSize	MessageStoreStats.SpillDispatcherRequestSize	Measures the number of operations on nonpersistent data dispatched for spilling to the data store.	6.0 and later	Per Messaging Engine	AverageStatistic	High
SpillDispatcherBatchSize	MessageStoreStats.SpillDispatcherBatchSize	Measures the batching of operations on nonpersistent data dispatched for spilling to the data store.	6.0 and later	Per Messaging Engine	AverageStatistic	High
SpillDispatcherAvoidanceCount	MessageStoreStats.SpillDispatcherAvoidanceCount	Measures the number of operations on nonpersistent data dispatched for spilling to the data store but whose spilling was subsequently unnecessary.	6.0 and later	Per Messaging Engine	AverageStatistic	High
SpillDispatcherAvoidanceSize	MessageStoreStats.SpillDispatcherAvoidanceSize	Measures the number of bytes associated with operations on nonpersistent data dispatched for spilling to the data store but whose spilling was subsequently unnecessary.	6.0 and later	Per Messaging Engine	AverageStatistic	High
PersistentDispatcherRequestSize	MessageStoreStats.PersistentDispatcherRequestSize	Measures the number of operations on reliable persistent data dispatched for writing to the data store.	6.0 and later	Per Messaging Engine	AverageStatistic	High
PersistentDispatcherBatchSize	MessageStoreStats.PersistentDispatcherBatchSize	Measures the batching of operations on reliable persistent data dispatched for writing to the data store.	6.0 and later	Per Messaging Engine	AverageStatistic	High
PersistentDispatcherCancellationCount	MessageStoreStats.PersistentDispatcherCancellationCount	Counts the number of global transaction completion phases whose operations cancelled out before being written to the data store.	6.0 and later	Per Messaging Engine	AverageStatistic	High
PersistentDispatcherAvoidanceCount	MessageStoreStats.PersistentDispatcherAvoidanceCount	Measures the number of operations on reliable persistent data dispatched for writing to the data store but whose writing was subsequently unnecessary.	6.0 and later	Per Messaging Engine	AverageStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache

PersistentDispatcherAvoidanceSize	MessageStoreStats.PersistentDispatcherAvoidanceSize	Measures the number of bytes associated with operations on reliable persistent data which were dispatched for writing to the data store but whose writing was subsequently unnecessary.	6.0 and later	Per Messaging Engine	AverageStatistic	High
JDBCOpenCount	MessageStoreStats.JDBCOpenCount	JDBC connections open	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCTransactionCompleteCount	MessageStoreStats.JDBCTransactionCompleteCount	JDBC local transactions completed	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCTransactionCompleteCount	MessageStoreStats.JDBCTransactionCompleteCount	JDBC local transactions completed	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCTransactionAbortCount	MessageStoreStats.JDBCTransactionAbortCount	JDBC local transactions aborted	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCTransactionTime	MessageStoreStats.JDBCTransactionTime	Total execution time of internal batches	6.0 and later	Per Messaging Engine	TimeStatistic	High
JDBCItemInsertCount	MessageStoreStats.JDBCItemInsertCount	JDBC Item table inserts	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCItemDeleteCount	MessageStoreStats.JDBCItemDeleteCount	JDBC Item table deletes	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCItemUpdateCount	MessageStoreStats.JDBCItemUpdateCount	JDBC Item table updates	6.0 and later	Per Messaging Engine	CountStatistic	High
ItemInsertBatchCount	MessageStoreStats.ItemInsertBatchCount	Item table insert batches	6.0 and later	Per Messaging Engine	CountStatistic	High
ItemDeleteBatchCount	MessageStoreStats.ItemDeleteBatchCount	Item table delete batches	6.0 and later	Per Messaging Engine	CountStatistic	High
ItemUpdateBatchCount	MessageStoreStats.ItemUpdateBatchCount	Item table update batches	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCTransactionInsertCount	MessageStoreStats.JDBCTransactionInsertCount	JDBC transaction table inserts	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCTransactionDeleteCount	MessageStoreStats.JDBCTransactionDeleteCount	JDBC transaction table deletes	6.0 and later	Per Messaging Engine	CountStatistic	High
JDBCTransactionUpdateCount	MessageStoreStats.JDBCTransactionUpdateCount	JDBC transaction table updates	6.0 and later	Per Messaging Engine	CountStatistic	High
TransactionInsertBatchCount	MessageStoreStats.TransactionInsertBatchCount	Transaction table insert batches	6.0 and later	Per Messaging Engine	CountStatistic	High
TransactionDeleteBatchCount	MessageStoreStats.TransactionDeleteBatchCount	Transaction table delete batches	6.0 and later	Per Messaging Engine	CountStatistic	High
TransactionUpdateBatchCount	MessageStoreStats.TransactionUpdateBatchCount	Transaction table update batches	6.0 and later	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Expiry

Name	Key	Description	Version	Granularity	Type	Level
ExpiryIndexItemCount	MessageStoreStats.ExpiryIndexItemCount	Current number of items in the expiry index. These are items created with an expiry time in the future and which have not yet been consumed.	6.0 and later	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Transactions

Name	Key	Description	Version	Granularity	Type	Level
LocalTransactionStartCount	MessageStoreStats.LocalTransactionStartCount	Local transactions started	6.0 and later	Per Messaging Engine	CountStatistic	High
LocalTransactionAbortCount	MessageStoreStats.LocalTransactionAbortCount	Local transactions aborted	6.0 and later	Per Messaging Engine	CountStatistic	High
LocalTransactionCommitCount	MessageStoreStats.LocalTransactionCommitCount	Local transactions committed	6.0 and later	Per Messaging Engine	CountStatistic	High
GlobalTransactionStartCount	MessageStoreStats.GlobalTransactionStartCount	Global transactions started	6.0 and later	Per Messaging Engine	CountStatistic	High
GlobalTransactionInDoubtCount	MessageStoreStats.GlobalTransactionInDoubtCount	Global transactions in doubt	6.0 and later	Per Messaging Engine	CountStatistic	High
GlobalTransactionAbortCount	MessageStoreStats.GlobalTransactionAbortCount	Global transactions cancelled	6.0 and later	Per Messaging Engine	CountStatistic	High
GlobalTransactionCommitCount	MessageStoreStats.GlobalTransactionCommitCount	Global transactions committed	6.0 and later	Per Messaging Engine	CountStatistic	High

Mediation Framework Statistics:

Consult this information for properties of mediation framework statistics.

Counter definitions*:*

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Mediations > Mediation

Name	Key	Description	Version	Granularity	Type	Level
ThreadCount	Mediation.ThreadCount	The number of messages being mediated concurrently at a mediation.	6.0 and later	per mediation	RangeStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Mediations > Mediation > Destination

Name	Key	Description	Version	Granularity	Type	Level
MediationTime	Mediation.MediationTime	The amount of time in milliseconds taken to mediate a message at a mediated destination.	6.0 and later	per mediated destination	TimeStatistic	Low
MediatedMessagesCount	Mediation.MediatedMessageCount	The number of messages that have been mediated at a mediated destination.	6.0 and later	per mediated destination	CountStatistic	Low

Message Processor Statistics:

Consult this information for properties of message processor statistics.

Counter definitions*:*

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Queues

Name	Key	Description	Version	Granularity	Type	Level
AvailableMessageCount	QueueStats.AvailableMessageCount	The number of messages available for a queue for consumption. If this number is close to the destination high messages value then review the high messages value.	6.0 and later and later	Per destination	CountStatistic	Low
UnavailableMessageCount	QueueStats.UnavailableMessageCount	The number of messages locked or uncommitted, this means messages that have been added or removed but the transaction has not been committed yet. If this number is high then check which messages are locked and why.	6.0 and later and later	Per destination	CountStatistic	Low
LocalProducerAttaches	QueueStats.LocalProducerAttachesCount	The number of times an attachment has been made to this queue by local producers. The lifetime of this value is the lifetime of the messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
LocalProducerCount	QueueStats.LocalProducerCount	The number of currently attached local producers.	6.0 and later and later	Per destination	CountStatistic	Low
LocalConsumerAttaches	QueueStats.LocalConsumerAttachesCount	The number of times an attachment has been made to this queue by local consumers. The lifetime of this value is the lifetime of the messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
LocalConsumerCount	QueueStats.LocalConsumerCount	The number of currently attached local consumers.	6.0 and later and later	Per destination	CountStatistic	Low
TotalMessagesProduced	QueueStats.TotalMessagesProducedCount	The total number of messages produced to this queue, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Queues

BestEffortNon-persistentMessagesProduced	QueueStats.BestEffortNonPersistentMessagesProducedCount	The number of Best Effort Non-persistent messages produced, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
ExpressNon-persistentMessagesProduced	QueueStats.ExpressNonPersistentMessagesProducedCount	The number of Express Non-persistent messages produced, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
ReliableNon-persistentMessagesProduced	QueueStats.ReliableNonPersistentMessagesProducedCount	The number of Reliable Non-persistent messages produced, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
ReliablePersistentMessagesProduced	QueueStats.ReliablePersistentMessagesProducedCount	The number of Reliable Persistent messages produced, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
AssuredPersistentMessagesProduced	QueueStats.AssuredPersistentMessagesProducedCount	The number of Assured Persistent messages produced, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
TotalMessagesConsumed	QueueStats.TotalMessagesConsumedCount	The total number of messages consumed from this queue, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
BestEffortNon-persistentMessagesConsumed	QueueStats.BestEffortNonPersistentMessagesConsumedCount	The number of Best Effort Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
ExpressNon-persistentMessagesConsumed	QueueStats.ExpressNonPersistentMessagesConsumedCount	The number of Express Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Queues

ReliableNon-persistentMessagesConsumed	QueueStats.ReliableNonPersistentMessagesConsumedCount	The number of Reliable Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
ReliablePersistentMessagesConsumed	QueueStats.ReliablePersistentMessagesConsumedCount	The number of Reliable Persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
AssuredPersistentMessagesConsumed	QueueStats.AssuredPersistentMessagesConsumedCount	The number of Assured Persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
ReportEnabledMessagesExpired.	QueueStats.ReportEnabledMessagesExpiredCount	The number of report enabled messages that expired while on this queue.	6.0 and later and later	Per destination	CountStatistic	Low
AggregateMessageWaitTime	QueueStats.AggregateMessageWaitTime	The time spent by messages in the bus at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	6.0 and later and later	Per destination	CountStatistic	Low
LocalMessageWaitTime	QueueStats.LocalMessageWaitTime	The time spent by messages on this queue at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	6.0 and later and later	Per destination	CountStatistic	Low

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Queues

LocalOldestMessageAge	QueueStats.LocalOldestMessageAge	The longest time any message has spent on this queue. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	6.0 and later and later	Per destination	CountStatistic	Low

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Topicspaces

Name	Key	Description	Version	Granularity	Type	Level
IncompletePublicationCount	TopicspaceStats.IncompletePublicationCount	The number of publications not yet received by all current subscribers. If this number is unexpected then view the publication via the admin console to take any actions.	6.0 and later and later	Per destination	CountStatistic	Low
LocalPublisherAttaches	TopicspaceStats.LocalPublisherAttachesCount	The number of times an attachment has been made to this topicspace by local producers. The lifetime of this value is the lifetime of the messaging engine.	6.0 and later and later	Per destination	CountStatistic	Low
LocalPublisherCount	TopicspaceStats.LocalPublisherCount	The number of local publishers to topics in this topicspace.	6.0 and later and later	Per destination	CountStatistic	Low
TotalLocalSubscriptionCount	TopicspaceStats.TotalLocalSubscriptionCount	The number of local subscriptions to topics in this topicspace. Each subscription is counted once, even if the topic includes wildcards.	6.0 and later and later	Per destination	CountStatistic	Low
Non-DurableLocalSubscriptionCount	TopicspaceStats.NonDurableLocalSubscriptionCount	The number of non-durable subscriptions.	6.0 and later and later	Per destination	CountStatistic	Low
DurableLocalSubscriptioCount	TopicspaceStats.DurableLocalSubscriptionCount	The number of durable subscriptions.	6.0 and later and later	Per destination	CountStatistic	Low
TotalMessagesPublished	TopicspaceStats.TotalMessagesPublishedCount	The total number of publications to this topicspace.	6.0 and later and later	Per destination	CountStatistic	Low
BestEffortNon-persistentMessagesPublished	TopicspaceStats.BestEffortNonPersistentMessagesPublishedCount	The number of Best Effort Non- persistent messages published	6.0 and later and later	Per destination	CountStatistic	Low
ExpressNon-persistentMessagesPublished	TopicspaceStats.ExpressNonPersistentMessagesPublishedCount	The number of Express Non-persistent messages published	6.0 and later and later	Per destination	CountStatistic	Low

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Topicspaces

ReliableNon-persistentMessagesPublished	TopicspaceStats.ReliableNonPersistentMessagesPublishedCount	The number of Reliable Non-persistent messages published	6.0 and later and later	Per destination	CountStatistic	Low
ReliablePersistentMessagesPublished	TopicspaceStats.ReliablePersistentMessagesPublishedCount	The number of Reliable Persistent messages published	6.0 and later and later	Per destination	CountStatistic	Low
AssuredPersistentMessagesPublished	TopicspaceStats.AssuredPersistentMessagesPublishedCount	The number of Assured Persistent messages published	6.0 and later and later	Per destination	CountStatistic	Low
TotalLocalSubscriptionHits	TopicspaceStats.TotalLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched topicspace publications.	6.0 and later and later	Per destination	CountStatistic	Low
BestEffortNon-persistentLocalSubscriptionHits	TopicspaceStats.BestEffortNonPersistentLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched Best Effort Non-persistent publications.	6.0 and later and later	Per destination	CountStatistic	Low
ExpressNon-persistentLocal SubscriptionHits	TopicspaceStats.ExpressNonPersistentLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched Express Non-persistent publications.	6.0 and later and later	Per destination	CountStatistic	Low
ReliableNon-persistentLocalSubscriptionHits	TopicspaceStats.ReliableNonPersistentLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched Reliable Non-persistent publications.	6.0 and later and later	Per destination	CountStatistic	Low
ReliablePersistentLocalSubscriptionHits	TopicspaceStats.ReliablePersistentLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched Reliable Persistent publications.	6.0 and later and later	Per destination	CountStatistic	Low
AssuredPersistentLocalSubscriptionHits	TopicspaceStats.AssuredPersistentLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched Assured Persistent publications.	6.0 and later	Per destination	CountStatistic	Low
ReportEnabledPublicationExpired	TopicspaceStats.ReportEnabledPublicationsExpiredCount	The number of report enabled incomplete publications that expired while on this topicspace.	6.0 and later	Per destination	CountStatistic	Low

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Topicspaces

LocalOldestPublication	TopicspaceStats.LocalOldestPublicationAge	The longest time any publication has spent on this topicspace. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	6.0 and later	Per destination	TimeStatistic	max

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Topicspaces > Topic > DurableSubscriptions

Name	Key	Description	Version	Granularity	Type	Level
AvailabeMessageCount	DurableSubscriptionStats.AvailableMessageCount	The number of messages waiting to be consumed.	6.0 and later	per mediated destination	CountStatistic	Low
TotalMessagesConsumed	DurableSubscriptionStats.TotalMessagesConsumedCount	The total number of messages consumed from this durable subscription.	6.0 and later	per mediated destination	CountStatistic	Low
BestEffortNon-persistentMessagesConsumed	DurableSubscriptionStats.BestEffortNonPersistentMessagesConsumedCount	The number of Best Effort Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later	per mediated destination	CountStatistic	Low
ExpressNon-persistentMessagesConsumed	DurableSubscriptionStats.ExpressNonPersistentMessagesConsumedCount	The number of Express Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later	per mediated destination	CountStatistic	Low
ReliableNon-persistentMessagesConsumed	DurableSubscriptionStats.ReliableNonPersistentMessagesConsumedCount	The number of Reliable Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later	per mediated destination	CountStatistic	Low
ReliablePersistentMessages Consumed	DurableSubscriptionStats.ReliablePersistentMessagesConsumedCount	The number of Reliable Persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later	per mediated destination	CountStatistic	Low
AssuredPersistentMessages Consumed	DurableSubscriptionStats.AssuredPersistentMessagesConsumedCount	The number of Assured Persistent messages consumed, for the lifetime of this messaging engine.	6.0 and later	per mediated destination	CountStatistic	Low
AggregateMessageWaitTime	DurableSubscriptionStats.AggregateMessageWaitTime	The time spent by messages in the bus at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	6.0 and later	per mediated destination	TimeStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Topicspaces > Topic > DurableSubscriptions

LocalMessageWaitTime	DurableSubscriptionStats.LocalMessageWaitTime	The time spent by messages on this durable subscription at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	6.0 and later	per mediated destination	TimeStatistic	High
LocalOldestPublication	DurableSubscriptionStats.LocalOldestPublicationAge	The longest time any message has spent on this subscription. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	6.0 and later	per mediated destination	TimeStatistic	Max

Communications statistics:

Communications statistics is a type of data that PMI maintains within the entire WebSphere Application Server domain.

Counter definitions*:*

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

Name	Key	Description	Version	Granularity	Type	Level
BytesSentAtHighestPriorityCount	ClientDetailedStats.BytesSentAtHighestPriority	Number of bytes of data transmitted at the highest possible priority for transmission. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtVeryHighPriorityCount	ClientDetailedStats.BytesSentAtVeryHighPriority	Number of bytes of data transmitted at a very high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtHighPriorityCount	ClientDetailedStats.BytesSentAtHighPriority	Number of bytes of data transmitted at a high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS9PriorityCount	ClientDetailedStats.BytesSentAtJMS9Priority	Number of bytes of data transmitted at the priority used by JMS priority 9 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesSentAtJMS8PriorityCount	ClientDetailedStats.BytesSentAtJMS8Priority	Number of bytes of data transmitted at the priority used by JMS priority 8 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS7PriorityCount	ClientDetailedStats.BytesSentAtJMS7Priority	Number of bytes of data transmitted at the priority used by JMS priority 7 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS6PriorityCount	ClientDetailedStats.BytesSentAtJMS6Priority	Number of bytes of data transmitted at the priority used by JMS priority 6 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesSentAtJMS5PriorityCount	ClientDetailedStats.BytesSentAtJMS5Priority	Number of bytes of data transmitted at the priority used by JMS priority 5 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS4PriorityCount	ClientDetailedStats.BytesSentAtJMS4Priority	Number of bytes of data transmitted at the priority used by JMS priority 4 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS3PriorityCount	ClientDetailedStats.BytesSentAtJMS3Priority	Number of bytes of data transmitted at the priority used by JMS priority 3 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesSentAtJMS2PriorityCount	ClientDetailedStats.BytesSentAtJMS2Priority	Number of bytes of data transmitted at the priority used by JMS priority 2 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS1PriorityCount	ClientDetailedStats.BytesSentAtJMS1Priority	Number of bytes of data transmitted at the priority used by JMS priority 1 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS0PriorityCount	ClientDetailedStats.BytesSentAtJMS0Priority	Number of bytes of data transmitted at the priority used by JMS priority 0 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesSentAtLowPriorityCount	ClientDetailedStats.BytesSentAtLowPriority	Number of bytes of data transmitted at a low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtVeryLowPriorityCount	ClientDetailedStats.BytesSentAtVeryLowPriority	Number of bytes of data transmitted at a very low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtLowestPriorityCount	ClientDetailedStats.BytesSentAtLowestPriority	Number of bytes of data transmitted at the lowest possible priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtHighestPriorityCount	ClientDetailedStats.BytesReceivedAtHighestPriority	Number of bytes of data received at the highest possible priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtVeryHighPriorityCount	ClientDetailedStats.BytesReceivedAtVeryHighPriority	Number of bytes of data received at a very high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesReceivedAtHighPriorityCount	ClientDetailedStats.BytesReceivedAtHighPriority	Number of bytes of data received at a high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS9PriorityCount	ClientDetailedStats.BytesReceivedAtJMS9Priority	Number of bytes of data received at the priority used by JMS priority 9 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS8PriorityCount	ClientDetailedStats.BytesReceivedAtJMS8Priority	Number of bytes of data received at the priority used by JMS priority 8 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesReceivedAtJMS7PriorityCount	ClientDetailedStats.BytesReceivedAtJMS7Priority	Number of bytes of data received at the priority used by JMS priority 7 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS6PriorityCount	ClientDetailedStats.BytesReceivedAtJMS6Priority	Number of bytes of data received at the priority used by JMS priority 6 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS5PriorityCount	ClientDetailedStats.BytesReceivedAtJMS5Priority	Number of bytes of data received at the priority used by JMS priority 5 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesReceivedAtJMS4PriorityCount	ClientDetailedStats.BytesReceivedAtJMS4Priority	Number of bytes of data received at the priority used by JMS priority 4 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS3PriorityCount	ClientDetailedStats.BytesReceivedAtJMS3Priority	Number of bytes of data received at the priority used by JMS priority 3 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS2PriorityCount	ClientDetailedStats.BytesReceivedAtJMS2Priority	Number of bytes of data received at the priority used by JMS priority 2 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesReceivedAtJMS1PriorityCount	ClientDetailedStats.BytesReceivedAtJMS1Priority	Number of bytes of data received at the priority used by JMS priority 1 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS0PriorityCount	ClientDetailedStats.BytesReceivedAtJMS0Priority	Number of bytes of data received at the priority used by JMS priority 0 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtLowPriorityCount	ClientDetailedStats.BytesReceivedAtLowPriority	Number of bytes of data received at a low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtVeryLowPriorityCount	ClientDetailedStats.BytesReceivedAtVeryLowPriority	Number of bytes of data received at a very low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

BytesReceivedAtLowestPriorityCount	ClientDetailedStats.BytesReceivedAtLowestPriority	Number of bytes of data received at the lowest possible priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS9PriorityCount	ClientDetailedStats.MessagesSentAtJMS9Priority	Number of messages transmitted at JMS priority 9.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS8PriorityCount	ClientDetailedStats.MessagesSentAtJMS8Priority	Number of messages transmitted at JMS priority 8.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS7PriorityCount	ClientDetailedStats.MessagesSentAtJMS7Priority	Number of messages transmitted at JMS priority 7.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS6PriorityCount	ClientDetailedStats.MessagesSentAtJMS6Priority	Number of messages transmitted at JMS priority 6.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS5PriorityCount	ClientDetailedStats.MessagesSentAtJMS5Priority	Number of messages transmitted at JMS priority 5.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS4PriorityCount	ClientDetailedStats.MessagesSentAtJMS4Priority	Number of messages transmitted at JMS priority 4.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS3PriorityCount	ClientDetailedStats.MessagesSentAtJMS3Priority	Number of messages transmitted at JMS priority 3.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

MessagesSentAtJMS2PriorityCount	ClientDetailedStats.MessagesSentAtJMS2Priority	Number of messages transmitted at JMS priority 2.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS1PriorityCount	ClientDetailedStats.MessagesSentAtJMS1Priority	Number of messages transmitted at JMS priority 1.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS0PriorityCount	ClientDetailedStats.MessagesSentAtJMS0Priority	Number of messages transmitted at JMS priority 0.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS9PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS9Priority	Number of messages received at JMS priority 9.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS8PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS8Priority	Number of messages received at JMS priority 8.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS7PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS7Priority	Number of messages received at JMS priority 7.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS6PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS6Priority	Number of messages received at JMS priority 6.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS5PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS5Priority	Number of messages received at JMS priority 5.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS4PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS4Priority	Number of messages received at JMS priority 4.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

MessagesReceivedAtJMS3PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS3Priority	Number of messages received at JMS priority 3.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS2PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS2Priority	Number of messages received at JMS priority 2.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS1PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS1Priority	Number of messages received at JMS priority 1.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS0PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS0Priority	Number of messages received at JMS priority 0.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Standard Statistics

Name	Key	Description	Version	Granularity	Type	Level
ClientsAttachedCount	ClientStats.ClientsAttached	The number of distinct client processes currently network connected to this application server.	6.0 and later	current clients/connections connected to this application server	CountStatistic	Low
APIConnectionsCount	ClientStats.APIConnections	The number of API sessions being used by clients that are currently network connected to this application server. Some of these API connections might be being by internal system processes on behalf of a client.	6.0 and later	current clients/connections connected to this application server	CountStatistic	Low
ErrorsCount	ClientStats.Errors	Communication errors that have occurred and resulted in a network connection to a client being disconnected.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
WritesCount	ClientStats.Writes	Number of write operations used to transmit data to client processes via network connections.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
ReadsCount	ClientStats.Reads	Number of read operations used to receive data from client processes via network connections.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
WritesBlockedCount	ClientStats.WritesBlocked	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with client processes.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
ReadsBlockedCount	ClientStats.ReadsBlocked	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with client processes.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MulticastWriteBytesCount	ClientStats.MulticastWriteBytes	Number of bytes transmitted using multicast protocols.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MulticastSendMessageCount	ClientStats.MulticastSendMessage	Number of messages transmitted using multicast protocols.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Clients > Standard Statistics

BufferedWriteBytesCount	ClientStats.BufferedWriteBytes	Number of bytes of data being held pending transmission. Large values might indicate network congestion or clients which are unable to process data fast enough to keep up with the application server.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
BufferedReadBytesCount	ClientStats.BufferedReadBytes	Number of bytes of data that have been received from the network and are held pending further processing. Large values might indicate that the application server is unable to process data fast enough to keep up with the clients attached.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesBytesWrittenCount	ClientStats.MessageBytesWritten	Number of bytes of message data sent to client processes over network connections. This does not include data used to negotiate the transmission of messages	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
MessageBytesReadCount	ClientStats.MessageBytesRead	Number of bytes of message data received from client processes over network connections. This does not include data used to negotiate the transmission of messages	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
TotalBytesWrittenCount	ClientStats.TotalBytesWritten	Number of bytes of data sent to client processes. This includes both message data and data used to negotiate the transmission of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low
TotalBytesReadCount	ClientStats.TotalBytesRead	Number of bytes of data received from client processes. This includes both message data and data used to negotiate the transmission of messages.	6.0 and later	All clients connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > Messaging Engines > Standard Statistics

Name	Key	Description	Version	Granularity	Type	Level
MEAttachedCount	MEStats.MEAttached	The number of distinct application server processes hosting messaging engines currently network connected to this application server.	6.0 and later	current applications servers hosting messaging engines/connections connected to this application server.	CountStatistic	Low
APIConnectionsCount	MEStats.APIConnections	The number of sessions being used by messaging engines that are currently network connected to this application server.	6.0 and later	current applications servers hosting messaging engines/connections connected to this application server.	CountStatistic	Low
ErrorsCount	MEStats.Errors	Communication errors that have occurred and resulted in a network connection to a messaging engine being disconnected.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
WritesCount	MEStats.Writes	Number of write operations used to transmit data to application server processes hosting messaging engines via network connections.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
ReadsCount	MEStats.Reads	Number of read operations used to receive data from application server processes hosting messaging engines via network connections.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
WritesBlockedCount	MEStats.WritesBlocked	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with application server processes hosting messaging engines.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
ReadsBlockedCount	MEStats.ReadsBlocked	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with application server processes hosting messaging engines.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low

BufferedWriteBytesCount	MEStats.BufferedWriteBytes	Number of bytes of data being held pending transmission. Large values might indicate network congestion or application server processes hosting messaging engines which are unable to process data fast enough to keep up with the application server.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
BufferedReadBytesCount	MEStats.BufferedReadBytes	Number of bytes of data that have been received from the network and are held pending further processing. Large values might indicate that the application server is unable to process data fast enough to keep up with the other application server processes hosting messaging engines that it is network attached.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
MessagesBytesWrittenCount	MEStats.MessageBytesWritten	Number of bytes of message data sent to application server processes hosting messaging engines over network connections. This does not include data used to negotiate the transmission of messages	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
MessageBytesReadCount	MEStats.MessageBytesRead	Number of bytes of message data received from application server processes hosting messaging engines over network connections. This does not include data used to negotiate the transmission of messages	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
TotalBytesWrittenCount	MEStats.TotalBytesWritten	Number of bytes of data sent to application server processes hosting messaging engines.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
TotalBytesReadCount	MEStats.TotalBytesRead	Number of bytes of data received from application server processes hosting messaging engines.	6.0 and later	All messaging engines connected or that have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > WMQ Client Links > Standard Statistics

Name	Key	Description	Version	Granularity	Type	Level
BatchesSentCount	MQClientLinkStats.BatchesSent	Number of batches of messages sent to network attached MQJMS clients.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
MessagesSentCount	MQClientLinkStats.MessagesSent	Number of messages sent to network attached WMQ JMS clients.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
MessagesReceivedCount	MQClientLinkStats.MessagesReceived	Number of messages received from network attached WMQ JMS clients.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
BytesSentCount	MQClientLinkStats.BytesSent	Number of bytes of data sent to network attached WMQ JMS clients. This includes bytes of message data as well as bytes of data used to control the flow of messages.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
BytesReceivedCount	MQClientLinkStats.BytesReceived	Number of bytes of data received from network attached WMQ JMS clients. This includes bytes of message data as well as bytes of data used to control the flow of messages.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
APICallsServicedCount	MQClientLinkStats.APICallsServiced	The number of MQ API call requests serviced on behalf of WMQ JMS clients.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
CommsErrorsCount	MQClientLinkStats.CommsErrors	The number of errors that have cause connections to WMQ JMS clients to be dropped.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
ClientsAttachedCount	MQClientLinkStats.ClientsAttached	The current number of WMQ JMS clients attached to this application server.	6.0 and later	WMQ JMS clients that are currently attached.	CountStatistic	Low
WritesBlockedCount	MQClientLinkStats.WritesBlocked	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WMQ JMS clients.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > WMQ Client Links > Standard Statistics

ReadsBlockedCount	MQClientLinkStats.ReadsBlocked	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WMQ JMS clients.	6.0 and later	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > WMQ Links > Standard Statistics

Name	Key	Description	Version	Granularity	Type	Level
BatchesSentCount	MQLinkStats.BatchesSent	Number of batches of messages sent to network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
BatchesReceivedCount	MQLinkStats.BatchesReceived	Number of batches of messages received from network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
MessagesSentCount	MQLinkStats.MessagesSent	Number of messages sent to network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
MessagesReceivedCount	MQLinkStats.MessagesReceived	Number of messages received from network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
SenderBytesSentCount	MQLinkStats.SenderBytesSent	Number of bytes of data sent by sender channels to network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
SenderBytesReceivedCount	MQLinkStats.SenderBytesReceived	Number of bytes of data received by sender channels from network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
ReceiverBytesSentCount	MQLinkStats.ReceiverBytesSent	Number of bytes of data sent by receiver channels to network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
ReceiverBytesReceivedCount	MQLinkStats.ReceiverBytesReceived	Number of bytes of data received by receiver channels from network attached WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
ShortRetriesCount	MQLinkStats.ShortRetries	Number of short retries. This indicates the number of times channels were disconnected and could not be re-established for short periods of time.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
LongRetriesCount	MQLinkStats.LongRetries	Number of long retries. This indicates the number of times channels were disconnected and could not be re-established for longer periods of time.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
CommsErrorsCount	MQLinkStats.CommsErrors	Number of communication errors that resulted in a network connection to a WMQ Queue Manager being disconnected.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low

Performance Modules > SIB Service > SIB Communications > WMQ Links > Standard Statistics

QMAttachedCount	MQLinkStats.QMAttached	Total number of WMQ Queue Managers currently network attached to this application server	6.0 and later	WMQ Queue Managers that are currently attached	CountStatistic	Low
WritesBlockedCount	MQLinkStats.WritesBlocked	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
ReadsBlockedCount	MQLinkStats.ReadsBlocked	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WMQ Queue Managers.	6.0 and later	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low

PMI data collection

PMI data collection can occur in three different interfaces, Java Management Extension, Performance Servlet, or PMI client API (deprecated).

The PMI data can be collected using the following interfaces:

- Java Management Extension (JMX) interface (J2EE MBeans and WebSphere Application Server Perf MBean)
- Performance Servlet
- PMI client API (deprecated)

JMX Interface

JMX interface is part of the J2EE specification and the recommended way to gather WebSphere Application Server performance data. PMI data can be gathered from the J2EE managed object MBeans or the WebSphere Application Server PMI Perf MBean. While the J2EE MBeans provide performance data about the specific component, the Perf MBean acts as a gateway to the WebSphere Application Server PMI service, and provides access to the performance data for all the components.

Performance Servlet

Performance Servlet provides a way to use an HTTP request to query the PMI data for the entire WebSphere Application Server administrative domain. Since the servlet provides the performance data through HTTP, issues such as firewalls are trivial to resolve. The performance servlet outputs the PMI data as an XML document.

PMI client API (deprecated)

PMI client API provides a wrapper class to deliver PMI data to a client. This API was introduced in WebSphere Application Server, Version 4.0 and deprecated in V6.0. The PMI client API uses the JMX infrastructure and the Perf MBean to retrieve the PMI data. PMI client provides the data using a WebSphere Application Server-specific data structure that was introduced in V4.0.

Custom PMI API

You can create specific statistics to best meet your monitoring interests by using custom PMI API.

PMI can be extended using the Custom PMI API to create application specific statistics. For example, a stock trading application can use Custom PMI API to create business specific statistics like ″number of stock sell transactions″ and ″number of stock buy transactions″.

Note that PMI provides detailed performance data about various runtime and application components. Starting with WebSphere Application Server Version 6.0, PMI offers approximately 180 or more performance statistics. Before creating new statistics, it is important to make sure that the same data is not captured by PMI already.

With WebSphere PMI, application developers can add their own application-specific instrumentation. The Custom PMI API simplifies the process of ″PMI enabling″ an application by providing an easy to use API. The statistics created via the Custom PMI can be accessed via the standard PMI and JMX interfaces that are used by monitoring tools including the Tivoli Performance Viewer.

PMI instrumentation is based on the J2EE 1.4 standard. As a result, Custom PMI supports all the Statistic types (CountStatistic, TimeStatistic, RangeStatistic, and BoundedRangeStatistic) defined in the JSR-77 Performance Data Framework. Custom PMI does not support user-defined Statistic types.

What you need to know

PMI collects performance data on runtime applications and provides interfaces that allow external applications to monitor the performance data.

With server side PMI, application developers can add their own instrumentation to their applications to help monitor their own predefined performance metrics.

Key features of Custom PMI:

- Create a custom Stats/PMI (Stats is J2EE terminology) module using an XML template.
- Used by the application to instrument code.
- Statistics in the custom Stats module can be accessed via the standard PMI and JMX interfaces that are used by monitoring tools, including the Tivoli Performance Viewer.
- PMI instrumentation is based on the J2EE 1.4 standard. As a result, Custom PMI supports all the Statistic types (CountStatistic, TimeStatistic, RangeStatistic, and BoundedRangeStatistic) defined in the JSR-77 Performance Data Framework.
- Custom PMI does not support user-defined Statistic types.

PMI is for application server performance monitoring, and the data collected by PMI is used to tune the application server resources such as pools, queues and caches etc. Since performance instrumentation and statistics can have considerable impact on the application server performance, it is necessary that every statistic added via Custom PMI is relevant towards solving a performance problem. When the statistic to be added is designed, the issues like the following should be taken into consideration:

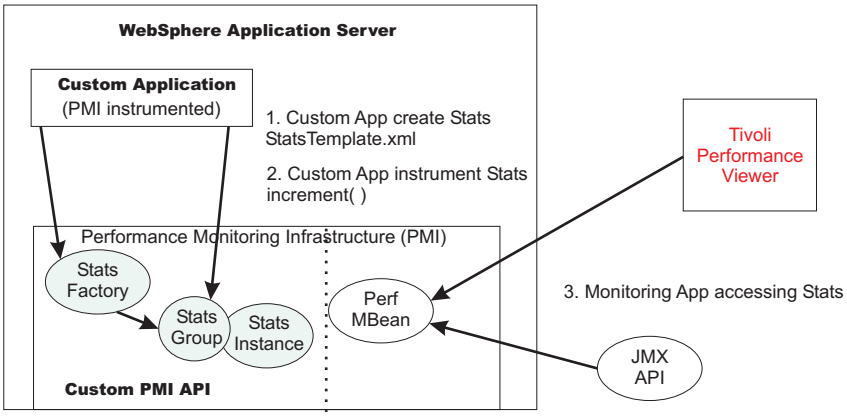
- Significance of the statistic with respect to solving performance problems.
- Relevance to tuning or configuring the application.
- Check for avoiding data redundancy and does not need frequent updates.

Custom PMI implementation - an example

The following diagram shows the custom PMI environment:

The following steps are required to instrument an application using Custom PMI:

1. Define a Stats module template. An XML document is used to define a set of statistics for a given application component. This XML document is used as a template to create the PMI data. The XML document should follow the DTD com/ibm/websphere/pmi/xml/stats.dtd
2. Create a Stats object using StatsFactory. The StatsFactory is used to create an instance (StatsInstance) or group (StatsGroup) of the Stats template. The StatsInstance object represents a single instance of the Stats template and contains all the statistics defined in the template. The StatsGroup is a logical collection of similar Stats instances. Custom PMI provides the flexibility to arrange the groups and instances in a tree structure.



The illustration above shows two instances of stock applications that are grouped under a StockAppStats group. The StockAppStats group can have multiple Stock applications, and each Stock application instance can have a StockBroker group. In this case, the StockAppStats group aggregates the statistics from StockApp1 and StockApp2, and the StockBroker group aggregates the statistics from all the StockBroker instances in their respective groups.

3. Instrument the application by updating the Stats object. To instrument, the application should call the Stats module for PMI service to maintain the raw counts. For example, to instrument the *number of sells* processed by the Stock application, create a Stats module with a statistic of type CountStatistic. When a sell transaction is processed, increment the *number of sells* statistic by calling: NumSellsCountStatistic.increment ();

Enabling PMI data collection

Enable PMI data collection to diagnose problems and tune application performance.

Performance Monitoring Infrastructure (PMI) needs to be enabled (by default PMI is enabled) before collecting any performance data. PMI should be enabled before the server starts. If PMI is enabled after the server is started, the server needs to be restarted to start the PMI.

When PMI service is enabled, the monitoring of individual components can be enabled or disabled dynamically. PMI provides four predefined statistic sets that can be used to enable a set of statistics. The following table provides details about the statistic sets. If the predefined statistic sets does not meet your monitoring requirement, the **Custom** option can be used to selectively enable or disable individual statistics.

Statistic set	Description
None	All statistics are disabled.
Basic	Statistics specified in J2EE 1.4, as well as top statistics like CPU usage and live HTTP sessions are enabled. This set is enabled *out-of-the-box* and provides basic performance data about runtime and application components.
Extended	Basic set plus key statistics from various WebSphere Application Server components like WLM and Dynamic caching are enabled. This set provides detailed performance data about various runtime and application components.
All	All statistics are enabled.
Custom	Enable or disable statistics selectively.

Custom setting

WebSphere Application Server Version 6.0 introduces fine-grained control to enable/disable statistics individually. The fine-grained control is available under the custom statistic set.

Though the Version 5.x monitoring levels {N, L, M, H, X} are deprecated in Version 6.0 and above, the 5.x PMI APIs are supported for backward compatibility. It is possible to use a Version 6.0 or later API to set the monitoring level and a Version 5.0 API to get the monitoring level. A new level 'F' – "fine-grained" is introduced to indicate to the Version 5.x API that the fine-grained or Version 6.x monitoring specification is in effect. This new level 'F' will be returned if a V5.x API is used to get the monitoring level from a server using Version 6.x monitoring specification.

In WebSphere Application Server Version 4.0 and Version 5.0, the statistics were enabled based on a monitoring/instrumentation level. The levels are None, Low, Medium, High and Max {N, L, M, H, X}. Enabling at a given level will include all the statistics at the given level plus counters from levels below the given level. So if you enable the Web Application module at level Medium {M} it enables all the counters at level M, plus all the Low {L} level counters. See "PMI data collection" on page 2210 for more information.

Sequential Update

In order to minimize the monitoring overhead, the updates to CountStatistic, AverageStatistic, and TimeStatistic are not synchronized. Since these statistic types track total and average, the extra accuracy is generally not worth the performance cost. The RangeStatistic and BoundedRangeStatistic are very

sensitive, therefore, they are always are synchronized. If needed, updates to all the statistic types can be synchronized by checking the **Use sequential update** check box.

Enable PMI using one of the following methods:

- "Enabling PMI using the administrative console"
- "Enabling PMI using the wsadmin tool" on page 2215
- "Enabling the Java virtual machine profiler data" on page 2221

Enabling PMI using the administrative console

When PMI service is enabled, the monitoring of individual components can be enabled or disabled dynamically.

To monitor performance data through the Performance Monitoring Infrastructure (PMI), you must first enable PMI through the administrative console.

1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the console navigation tree.
3. Click *server*.
4. Click the **Configuration** tab.
5. Click **Performance Monitoring Infrastructure** under Performance.
6. Select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
7. Optionally, select the check box **Use sequential update** to enable precise statistic update.
8. Optionally, choose a statistic set that needs to be monitored under **Currently Monitored Statistic Set**.
9. Optionally, click on **Custom** to selectively enable or disable statistics. Choose a component from the left side tree and enable or disable statistics on the right side table. Go back to the main PMI configuration page by clicking the **Performance Monitoring Infrastructure** link.
10. Click **Apply** or **OK**.
11. Click **Save**.
12. Restart the application server. The changes you make will not take effect until you restart the application server.

When in the **Configuration** tab, settings apply after the server is restarted.

Performance Monitoring Infrastructure settings

Use this page to specify settings for performance monitoring, including enabling performance monitoring, selecting the Performance Monitoring Infrastructure (PMI) module and setting monitoring levels.

To view this administrative console page, click **Servers** > **Application Servers** > *server* > **Performance Monitoring Infrastructure (PMI)**.

Enable Performance Monitoring Infrastructure (PMI):

Specifies whether the application server attempts to enable Performance Monitoring Infrastructure (PMI). If an application server is started when the PMI is disabled, you have to restart the server in order to enable it.

Use sequential counter updates:

Specifies whether access to PMI counters (for example, updates to the counters) is sequential. There is some performance overhead associated with enabling sequential update. The default setting is false.

Persist my changes:

Specifies whether changes to the runtime are also persisted for use on subsequent server startups, and not just the current server runtime. The default setting is false.

Selecting the Persist my changes option ensures that the counter settings are retained after you restart the server. This option itself does not remain selected the next time you come to the panel.

Currently monitored statistic set:

Specifies a pre-defined set of Performance Monitoring Infrastructure (PMI) statistics for all components in the server.

As part of fine-grained control feature, WebSphere Application Server provides new statistic sets, which are pre-defined, fixed server-side sets based on the PMI statistic usage scenarios. This enhancement allows a set of statistics to be enabled via a single action or call. The following statistic sets are defined:

None All statistics are disabled.

Basic Provides basic monitoring for application server resources and applications. This includes J2EE components, HTTP session information, CPU usage information, and the top 38 statistics. This is the default setting.

Extended
Provides extended monitoring, including the basic level of monitoring plus workload monitor, performance advisor, and Tivoli resource models. Extended provides key statistics from frequently used WebSphere Application Server components.

All Enables all statistics.

Custom
Provides fine-grained control with the ability to enable and disable individual statistics.

Custom monitoring level

Use this page to enable and disable specific PMI counters for individual PMI statistics.

To view this administrative console page, click **Servers** > **Application Servers** > *server* > **Performance Monitoring Infrastructure (PMI)** > **Custom**.

Custom monitoring level:

Click on the individual PMI statistic in the list on the left. The counters available for that module appear in the table on the right along with the counter type, a description of the counter, and its current status (Enabled or Disabled).

You can enable or disable each individual counter by selecting the counter and clicking the Enable or Disable button, respectively.

Custom monitoring level:

Click on the individual PMI module in the list on the left. The counters available for that statistic appear in the table on the right along with the counter type, a description of the counter, and its current status (Enabled or Disabled).

You can enable or disable each individual counter by selecting the counter and clicking the Enable or Disable button, respectively.

Note: Enabling or disabling a statistic set that contains subsets will affect the configuration of all the counters contained in the subsets. The parent statistic set that is selected passes down the current

configuration set to its subsets, which can disable any counters that were individually selected within specific subsets. For best results, configure the parent statistic first, and then select specific counters within its subsets.

Performance Monitoring Infrastructure collection

Use this page to configure Performance Monitoring Infrastructure (PMI).

To view this administrative console page, click **Monitoring and Tuning** > **Performance Monitoring Infrastructure (PMI)** .

Name:

Specifies the logical name of the server.

The Name property is read only.

Node:

Specifies the name of the node holding the server.

The Node name property is read only.

Version:

Specifies the version of the WebSphere Application Server product on which the server runs.

The Version property is read only.

Enabling PMI using the wsadmin tool

Use this page to learn how to enable Performance Monitoring Infrastructure using command line instead of the administrative console.

You can use the command line to enable Performance Monitoring Infrastructure (PMI).

1. Enable PMI using the administrative console (see “Enabling PMI using the administrative console” on page 2213). Make sure to restart the application server.
2. Run the **wsadmin** command, as described in Starting the wsadmin scripting client. Using **wsadmin**, you can invoke operations on Perf Mbean to obtain the PMI data, set or obtain PMI monitoring levels, and enable data counters.

   Note: If PMI data are not enabled yet, you need to first enable PMI data by invoking setInstrumentationLevel operation on PerfMBean.

   The following operations in Perf MBean can be used in **wsadmin**:

```
/** Get performance data information for stats */
public void getConfig (ObjectName mbean);

/** Returns the current statistic set */
public void getStatisticSet ();

/** Enable PMI data using the pre-defined statistic sets.
    Valid values for the statistic set are "basic", "extended", "all", and "none" */
public void setStatisticSet (String statisticSet);

/** Returns the current custom set specification as a string */
public void getCustomSetString ();

/** Customizing PMI data that is enabled using fine-grained control.
    This method allows to enable or disable statistics selectively.
```

```
    The format of the custom set specification string is STATS_NAME=ID1,ID2,ID3 seperated by ':',
  where STATS_NAME and IDs are defined in WS*Stat interfaces in com.ibm.websphere.pmi.stat package.
    Use * to enable all the statistics in the given PMI module. For example, to enable all the statistics
  for JVM and active count, pool size for thread pool use: jvmRuntimeModule=*:threadPoolModule=3,4.
    The string jvmRuntimeModule is the value of the constant WSJVMStats.NAME and threadPoolModule is the
value of WSThreadPoolStats.NAME.
 */
public void setCustomSetString (String customSpec, Boolean recursive);

/** Get stats for an MBean*/
public void getStatsObject (ObjectName mbean, Boolean recursive);

/** Set instrumentation level using String format.
    This should be used by scripting for an easy String processing.
    The level STR is a list of moduleName=Level connected by ":".
    NOTE: This method is deprecated in Version 6.1
 */
public void setInstrumentationLevel(String levelStr, Boolean recursive);


/** Get instrumentation level in String for all the top level modules.
    This should be used by scripting for an easy String processing.
    NOTE: This method is deprecated in Version 6.1
 */
public String getInstrumentationLevelString();

/** Return the PMI data in String
    NOTE: This method is deprecated in Version 6.1
 */
public String getStatsString(ObjectName on, Boolean recursive);

/** Return the PMI data in String
    Used for PMI modules/submodules without direct MBean mappings.
    NOTE: This method is deprecated in Version 6.1
 */
public String getStatsString(ObjectName on, String submoduleName, Boolean recursive);

/** Return the submodule names if any for the MBean
    NOTE: This method is deprecated in Version 6.1
 */
public String listStatMemberNames(ObjectName on);
```

If an MBean is a StatisticProvider, and if you pass its ObjectName to getStatsObject, you get the Statistic data for that MBean. MBeans with the following MBean types are statistic providers:

- DynaCache
- EJBModule
- EntityBean
- JDBCProvider
- J2CResourceAdapter
- JVM
- MessageDrivenBean
- ORB
- Server
- SessionManager
- StatefulSessionBean
- StatelessSessionBean
- SystemMetrics
- ThreadPool
- TransactionService
- WebModule
- Servlet
- WLMAppServer
- WebServicesService

- WSGW

The following are sample commands in **wsadmin** you can use to obtain PMI data:

Obtain the Perf MBean ObjectName

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
```

Invoke getStatisticSet operation

Use this method to find the statistic set that is currently in effect:

```
Wsadmin> $AdminControl invoke $perfName getStatisticSet
```

This method returns one of the following values: basic, extended, all, none.

Invoke setStatisticSet operation

Use this method to enable monitoring using a statistic set.

The valid statistic set values are: basic, extended, all, none.

```
Wsadmin> set params [java::new {java.lang.Object[]} 1]
Wsadmin> $params set 0 [java::new java.lang.String extended]
Wsadmin> set sigs  [java::new {java.lang.String[]} 1]
Wsadmin> $sigs set 0 java.lang.String
Wsadmin> $AdminControl invoke_jmx $perfOName setStatisticSet
$params $sigs
```

Invoke getConfig operation

Use this method to find information about the statistics for a given component.

```
Wsadmin> set jvmName [$AdminControl completeObjectName type=JVM,*]

Wsadmin> set params [java::new {java.lang.Object[]} 1]
Wsadmin> $params set 0 [java::new javax.management.ObjectName $jvmName]
Wsadmin> set sigs  [java::new {java.lang.String[]} 1]
Wsadmin> $sigs set 0 javax.management.ObjectName

Wsadmin> $AdminControl invoke_jmx $perfOName getConfig $params
$sigs
```

This method returns the following:

```
    Stats type=jvmRuntimeModule, Description=The performance data from
the Java virtual machine run time.

    {name=UpTime, ID=4, type=CountStatistic, description=The amount of
time (in seconds) that the Java virtual machine has been running.,
unit=SECOND, level=low, statisticSet=basic, resettable=false,
aggregatable=true}

    {name=UsedMemory, ID=3, type=CountStatistic, description=The amount
of used memory (in KBytes) in the Java virtual machine run time.,
unit=KILOBYTE, level=low,
    statisticSet=basic, resettable=false, aggregatable=true}

    {name=FreeMemory, ID=2, type=CountStatistic, description=The free
memory (in KBytes) in the Java virtual machine run time.,
unit=KILOBYTE, level=low, statisticSet=all, resettable=false,
aggregatable=true}
```

```
    {name=HeapSize, ID=1, type=BoundedRangeStatistic, description=The
total memory (in KBytes) in the Java virtual machine run time.,
unit=KILOBYTE, level=high, statisticSet=basic, resettable=false,
aggregatable=true}
```

Invoke getCustomSetString operation

This operation provides the current monitoring specification in a string format:

```
Wsadmin> $AdminControl invoke $perfName getCustomSetString
```

The output looks similar to the following:

```
    jvmRuntimeModule=4,3,1:systemModule=2,1:threadPoolModule=4,3:thread
PoolModule>HAManager.thread.pool=4,3:threadPoolModule>MessageListenerTh
readPool=4,3:threadPoolModule>ORB.thread.pool=4,3:threadPoolModule>Serv
let.Engine.Transports=4,3:threadPoolModule>TCS_DEFAULT=4,3:transactionM
odule=4,19,16,18,3,7,6,1,14
```

This output indicates that statistic ID's 1, 3, and 4 are enabled in the JVM component. The description of the statistic IDs can be found using the above getConfig operation or using the API documentation. The output contains the current monitoring specification for the entire server. The individual modules are separated by a **:**, and **>** is used as a separator within the module.

Invoke setCustomString operation

This operation can be used to enable or disable statistics selectively. In the following command the statistic IDs 1, 2, 3, and 4 are enabled for the JVM module. To enable all the statistic IDs use an asterisk (*****).

```
    Wsadmin> set params [java::new {java.lang.Object[]} 2]
    Wsadmin> $params set 0 [java::new java.lang.String
jvmRuntimeModule=1,2,3,4]
    Wsadmin> $params set 1 [java::new java.lang.Boolean false]

    Wsadmin> set sigs  [java::new {java.lang.String[]} 2]
    Wsadmin> $sigs set 0 java.lang.String
    Wsadmin> $sigs set 1 java.lang.Boolean


    Wsadmin> $AdminControl invoke_jmx $perfOName setCustomSetString
$params $sigs
```

Invoke getStatsObject operation

This operation is used to get the statistics for a given MBean. The following example gets the statistics for the JVM:

```
    Wsadmin> set jvmName [$AdminControl completeObjectName type=JVM,*]
    Wsadmin> set params [java::new {java.lang.Object[]} 2]
    Wsadmin> $params set 0 [java::new javax.management.ObjectName
$jvmName]
    Wsadmin> $params set 1 [java::new java.lang.Boolean false]
    Wsadmin> set sigs  [java::new {java.lang.String[]} 2]
    Wsadmin> $sigs set 0 javax.management.ObjectName
    Wsadmin> $sigs set 1 java.lang.Boolean
    Wsadmin> $AdminControl invoke_jmx $perfOName getStatsObject $params
$sigs
    Stats name=jvmRuntimeModule, type=jvmRuntimeModule#
    {
    name=HeapSize, ID=1, description=The total memory (in KBytes) in
the Java virtual machine run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=51200,
highWaterMark=263038, current=263038, integral=2.494158617766E12, lowerBound
=51200, upperBound=262144
```

```
    name=FreeMemory, ID=2, description=The free memory (in KBytes) in
the Java virtual machine run time., unit=KILOBYTE, type=CountStatistic,
count=53509

    name=UsedMemory, ID=3, description=The amount of used memory (in KBytes) in
the Java virtual machine run time., unit=KILOBYTE,
type=CountStatistic, count=209528

    name=UpTime, ID=4, description=The amount of time (in seconds) that
the Java virtual machine has been running., unit=SECOND,
type=CountStatistic, count=83050
}
```

Invoke getInstrumentationLevelString operation

Use *invoke*, because it has no parameter.

```
wsadmin>$AdminControl invoke $perfName
getInstrumentationLevelString
```

This command returns the following:

```
beanModule=H:cacheModule=H:connectionPoolModule=H:j2cModule=H:jvmRu
ntimeModule=H:orbPerfModule=H:servletSessionsModule=H:systemModule=
H:threadPoolModule=H:transactionModule=H:webAppModule=H
```

Note: You can change the level (n, l, m, h, x) in the above string and then pass it to setInstrumentationLevel method.

Invoke setInstrumentationLevel operation - enable/disable PMI counters

- sSet parameters (″pmi=l″ is the simple way to set all modules to the low level).

  ```
  wsadmin>set params [java::new {java.lang.Object[]} 2]

  wsadmin>$params set 0 [java::new java.lang.String pmi=l]
      wsadmin>$params set 1 [java::new java.lang.Boolean true]
  ```

- Set signatures.

  ```
  wsadmin>set sigs  [java::new {java.lang.String[]} 2]

  wsadmin>$sigs set 0 java.lang.String
      wsadmin>$sigs set 1 java.lang.Boolean
  ```

- Invoke the method. Use **invoke_jmx**, because it has a parameter.

  ```
  wsadmin>$AdminControl invoke_jmx $perfOName
  setInstrumentationLevel $params $sigs
  ```

This command does not return anything.

Note: The PMI level string can be as simple as *pmi=level* (where level is n, l, m, h, or x), or something like *module1=level1:module2=level2:module3=level3* with the same format shown in the string returned from `getInstrumentationLevelString`.

Invoke getStatsString(ObjectName, Boolean) operation

If you know the MBean ObjectName, you can invoke the method by passing the right parameters. As an example, JVM MBean is used here.

- Get MBean query string. For example, JVM MBean.

  ```
   wsadmin>set jvmName [$AdminControl completeObjectName
  type=JVM,*]
  ```

- Set parameters.

  ```
  wsadmin>set params [java::new {java.lang.Object[]} 2]

  wsadmin>$params set 0 [$AdminControl makeObjectName $jvmName]
  wsadmin>$params set 1 [java::new java.lang.Boolean true]
  ```

- Set signatures.

```
wsadmin>set sigs  [java::new {java.lang.String[]} 2]
    wsadmin>$sigs set 0 javax.management.ObjectName wsadmin>$sigs
    set 1 java.lang.Boolean
```

- Invoke method.

```
 wsadmin>$AdminControl invoke_jmx $perfOName getStatsString
$params $sigs
```

This command returns the following:

```
{Description jvmRuntimeModule.desc} {Descriptor {{Node wenjianpc}
{Server server
1} {Module jvmRuntimeModule} {Name jvmRuntimeModule} {Type
MODULE}}} {Level 7} {
Data {{{Id 4} {Descriptor {{Node wenjianpc} {Server server1}
{Module jvmRuntimeM
odule} {Name jvmRuntimeModule} {Type DATA}}} {PmiDataInfo {{Name
jvmRuntimeModul
e.upTime} {Id 4} {Description jvmRuntimeModule.upTime.desc} {Level
1} {Comment {
The amount of time in seconds the JVM has been running}}
{SubmoduleName null} {T
ype 2} {Unit unit.second} {Resettable false}}} {Time 1033670422282}
{Value {Coun
t 638} }} {{Id 3} {Descriptor {{Node wenjianpc} {Server server1}
{Module jvmRunt
imeModule} {Name jvmRuntimeModule} {Type DATA}}} {PmiDataInfo
{{Name jvmRuntimeM
odule.usedMemory} {Id 3} {Description
jvmRuntimeModule.usedMemory.desc} {Level 1
} {Comment {Used memory in JVM runtime}} {SubmoduleName null} {Type
2} {Unit uni
t.kbyte} {Resettable false}}} {Time 1033670422282} {Value {Count
66239} }} {{Id
2} {Descriptor {{Node wenjianpc} {Server server1} {Module
jvmRuntimeModule} {Nam
e jvmRuntimeModule} {Type DATA}}} {PmiDataInfo {{Name
jvmRuntimeModule.freeMemor
y} {Id 2} {Description jvmRuntimeModule.freeMemory.desc} {Level 1}
{Comment {Fre
e memory in JVM runtime}} {SubmoduleName null} {Type 2} {Unit
unit.kbyte} {Reset
table false}}} {Time 1033670422282} {Value {Count 34356} }} {{Id 1}
{Descriptor
{{Node wenjianpc} {Server server1} {Module jvmRuntimeModule} {Name
jvmRuntimeMod
ule} {Type DATA}}} {PmiDataInfo {{Name
jvmRuntimeModule.totalMemory} {Id 1} {Des
cription jvmRuntimeModule.totalMemory.desc} {Level 7} {Comment
{Total memory in
JVM runtime}} {SubmoduleName null} {Type 5} {Unit unit.kbyte}
{Resettable false}
}} {Time 1033670422282} {Value {Current 100596} {LowWaterMark
38140} {HighWaterM
ark 100596} {MBean 38140.0} }}}}
```

Invoke getStatsString (ObjectName, String, Boolean) operation

This operation takes an additional String parameter, and it is used for PMI modules that do not have matching MBeans. In this case, the parent MBean is used with a String name representing the PMI module. The String names available in an MBean can be found by invoking listStatMemberNames. For example, *beanModule* is a logic module aggregating PMI data over all Enterprise JavaBeans, but there is no MBean for beanModule. Therefore, you can pass server MBean ObjectName and a String (*beanModule*) to get PMI data in beanModule.

- Get MBean query string. For example, server MBean

```
wsadmin>set mySrvName [$AdminControl completeObjectName
    type=Server,name=server1,
    node=wenjianpc,*]
```

- Set parameters.

```
wsadmin>set params [java::new {java.lang.Object[]} 3]

wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]

wsadmin>$params set 1 [java::new java.lang.String beanModule]
    wsadmin>$params set 2 [java::new java.lang.Boolean true]
```

- Set signatures.

```
wsadmin>set sigs  [java::new {java.lang.String[]} 3]

wsadmin>$sigs set 0 javax.management.ObjectName

wsadmin>$sigs set 1 java.lang.String
    wsadmin>$sigs set 2 java.lang.Boolean
```

- Invoke method.

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString
$params $sigs
```

This command returns PMI data in all the Enterprise JavaBeans within the BeanModule hierarchy because the recursive flag is set to true.

Note: This method is used to get stats data for the PMI modules that do not have direct MBean mappings.

Invoke listStatMemberNames operation

- Get MBean queryString. For example, Server.

```
wsadmin>set mySrvName [$AdminControl completeObjectName
    type=Server,name=server1,

node=wenjianpc,*]
```

- Set parameter.

```
wsadmin>set params [java::new {java.lang.Object[]} 1]
wsadmin>$params set 0 [$AdminControl makeObjectName
$mySrvName]
```

- Set signatures.

```
wsadmin>set sigs  [java::new {java.lang.String[]} 1]
wsadmin>$sigs set 0 javax.management.ObjectName
    wsadmin>$AdminControlinvoke_jmx $perfOName
    listStatMemberNames $params $sigs
```

This command returns the PMI module and submodule names, which have no direct MBean mapping. The names are seperated by a space ″ ″. You can then use the name as the String parameter in getStatsString method. For example:

```
beanModule connectionPoolModule j2cModule servletSessionsModule
threadPoolModule
webAppModule
```

Enabling the Java virtual machine profiler data

Use the Java virtual machine profiler to enable the collection of information.

Use the Java Virtual Machine Tool Interface (JVMTI) to collect data about garbage collection and thread states from the JVM that runs the application server. When JVMTI is enabled, a collection of JVMTI-specific statistics can be enabled through the predefined All statistic set or through the use of a Custom statistic set.

To enable JVMTI data reporting for each individual application server:

1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the administrative console navigation tree.

3. Click the application server for the JVM profiler that needs enabling.
4. Click **Java and Process Management** under Server Infrastructure.
5. Click **Process Definition**.
6. Click **Java virtual machine** under Additional properties.
7. Type *-agentlib:pmiJvmtiProfiler* in the **Generic JVM arguments** field.

   Note: If the deprecated JVMPI profiler is used in WebSphere Application Server Version 6.1 type *-XrunpmiJvmpiProfiler* in the **Generic JVM arguments** field. For more information, refer to "Java virtual machine profiling."
8. Click **Apply** or **OK**.
9. Click **Save**.
10. Click **Servers** > **Application Servers** in the administrative console navigation tree.
11. Click the application server for the JVM profiler that needs enabling.
12. Click the **Configuration** tab. When in the **Configuration** tab, settings apply when the server is restarted. When in the **Runtime** Tab, settings apply immediately. Performance Monitoring Infrastructure (PMI) can be enabled only in the **Configuration** tab.
13. Click **Performance Monitoring Infrastructure** under Performance.
14. Select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
15. Click **Custom** and select **JVM Runtime** on the left-side tree.
16. Click a JVM profiler group under **JVM Runtime** and enable or disable the statistic on the right-side table. Go back to the main PMI configuration page, by clicking PMI.
17. Click **Apply** or **OK**.
18. Click **Save**.
19. Start the application server, or restart the application server if it is currently running.
20. Refresh the Tivoli Performance Viewer if you are using it. The changes you make will not take affect until you restart the application server.

Java virtual machine profiling

Use Java virtual machine profiling to gather data about your system for performance analysis.

The Java virtual machine Tool Interface (JVMTI) is a native programming interface that provides tools the ability to inspect the state of the Java virtual machine (JVM). This interface is new for the JVM, V1.5. JVMTI replaces the Java virtual machine Profiling Interface (JVMPI), which is supported in WebSphere Application Server, Version 6.0.2 and earlier. The JVMPI interface is deprecated as of WebSphere Application Server Version 6.1.

Both interfaces (JVMTI and JVMPI) provide the ability to collect information about the JVM that runs the application server. The Tivoli Performance Viewer leverages these interfaces to enable more comprehensive performance analysis.

JVMTI is a two-way function call interface between the JVM and an in-process profiler agent. The JVM notifies the profiler agent of various events, for example, garbage collection and thread starts. The profiler agent activates or deactivates specific event notifications that are based on the needs of the profiler. The allocate, move, and free object counters that are provided through the JVMPI interface are not available through the JVMTI implementation in WebSphere Application Server Version 6.1.

JVMTI supports partial profiling, by enabling you to choose which types of profiling information to collect and to select certain subsets of the time during which the JVM API is active. JVMTI moderately increases the performance impact. Therefore, it is recommended that you use JVMTI monitoring to help diagnose application problems only.

Developing your own monitoring applications

You can use the Performance Monitoring Infrastructure (PMI) interfaces to develop your own applications to collect and display performance information.

There are three such interfaces - a Java Machine Extension (JMX)-based interface, a PMI client interface, and a servlet interface. All three interfaces return the same underlying data.

The JMX interface is accessible through the WebSphere Application Server administartive client as described in “Using the JMX interface to develop your own monitoring application” on page 2239. The PMI client interface is a Java interface that works with Version 3.5.5 and above. The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.

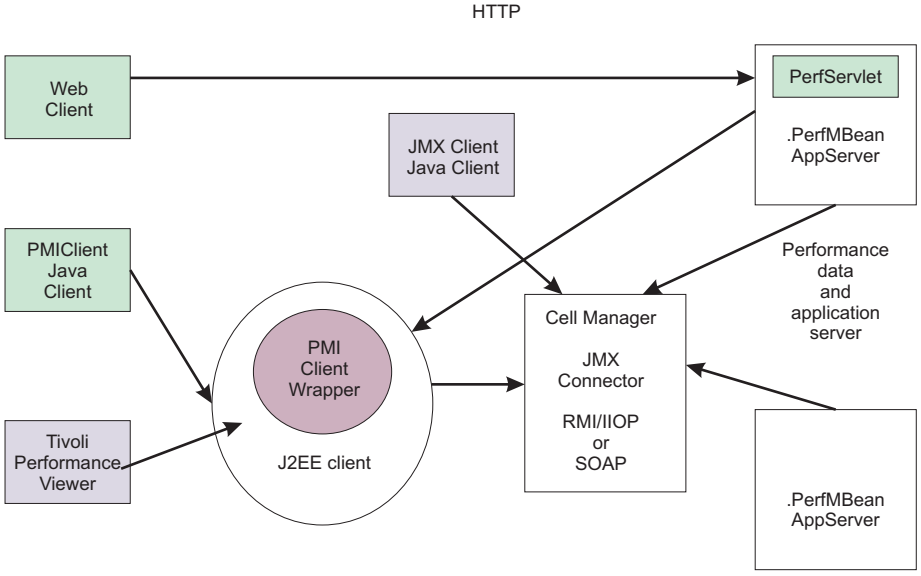
Note: The PMI client interface is deprecated in Version 6.1. The JMX interface is the recommended way to collect PMI data.

1. “Using PMI client to develop your monitoring application (deprecated)” on page 2224.
2. “Performance servlet (PerfServlet)” on page 2236
3. “Compiling your monitoring applications” on page 2257
4. “Running your new monitoring applications” on page 2257
5. “Using the JMX interface to develop your own monitoring application” on page 2239.
6. “Developing PMI interfaces (Version 4.0) (deprecated)” on page 2256.

PMI client interface (deprecated)

The data provided by the Performance Monitoring Infrastructure (PMI) client interface is documented here.

Access to the PMI client interface data is provided in a hierarchical structure. Descending from the object are node information objects, module information objects, CpdCollection objects and CpdData objects. Using Version 5.0, you will get Stats and Statistic objects. The node and server information objects contain no performance data, only static information.



Each time a client retrieves performance data from a server, the data is returned in a subset of this structure; the form of the subset depends on the data retrieved. You can update the entire structure with new data, or update only part of the tree, as needed.

The JMX statistic data model is supported, as well as the existing CPD data model from Version 4.0. When you retrieve performance data using the Version 5.0 PMI client API, you get the Stats object, which

includes Statistic objects and optional sub-Stats objects. When you use the Version 4.0 PMI client API to collect performance data, you get the CpdCollection object, which includes the CpdData objects and optional sub-CpdCollection objects.

The following are additional Performance Monitoring Infrastructure (PMI) interfaces:
- BoundaryStatistic
- BoundedRangeStatistic
- CountStatistic
- MBeanStatDescriptor
- MBeanLevelSpec
- New Methods in PmiClient
- RangeStatistic
- Stats
- Statistic
- TimeStatistic

The following PMI interfaces introduced in Version 4.0 are also supported:
- CpdCollection
- CpdData
- CpdEventListener and CpdEvent
- CpdFamily class
- CpdValue
  - CpdLong
  - CpdStat
  - CpdLoad
- PerfDescriptor
- PmiClient class

The CpdLong maps to CountStatistic; CpdStat maps to Time Statistic; CpdCollection maps to Stats; and CpdLoad maps to RangeStatistic and BoundedRangeStatistic.

Note: Version 4.0 PmiClient APIs are supported in this version, however, there are some changes. The data hierarchy is changed in some PMI modules, notably the enterprise bean module and HTTP sessions module. If you have an existing PmiClient application, and you want to run it against Version 5.0, you might have to update the PerfDescriptor(s) based on the new PMI data hierarchy. Also, the getDataName and getDataId methods in PmiClient are changed to be non-static methods in order to support multiple WebSphere Application Server versions. You might have to update your existing application which uses these two methods.

Using PMI client to develop your monitoring application (deprecated)

You can use the Performance Monitoring Infrastructure (PMI) interfaces to develop your own applications to collect and display performance information.

The following is the programming model for Performance Monitoring Infrastructure (PMI) client:

1. Create an instance of PmiClient. This is used for all subsequent method calls.
2. **Optional:** You can create your own MBeans. Refer to Extending the WebSphere Application Server administrative system with custom MBeans.
3. Call the listNodes() and listServers(nodeName) methods to find all the nodes and servers in the WebSphere Application Server domain. The PMI client provides two sets of methods: one set in Version 5.0 and the other set inherited from Version 4.0. You can only use one set of methods. Do not mix them together.
4. Call listMBeans and listStatMembers to get all the available MBeans and MBeanStatDescriptors.
5. Call the getStats method to get the Stats object for the PMI data.

6. **Optional:** The client can also call setStatLevel or getStatLevel to set and get the monitoring level. Use the MBeanLevelSpec objects to set monitoring levels.

If you prefer to use the Version 4.0 interface, the model is essentially the same, but the object types are different:
1. Create an instance of PmiClient.
2. Call the listNodes() and listServers(nodeName) methods to find all the nodes and servers in the WebSphere Application Server domain.
3. Call listMembers to get all the perfDescriptor objects.
4. Use the PMI client's get or gets method to get CpdCollection objects. These contain snapshots of performance data from the server. The same structure is maintained and its update method is used to refresh the data.
5. (Optional) The client can also call setInstrumentationLevel or getInstrumentationLevel to set and get the monitoring level.

Performance Monitoring Infrastructure client (WebSphere Version 4.0)

A Performance Monitoring Infrastructure (PMI) client is an application that receives PMI data from servers and processes this data.

In WebSphere Application Server Version 4.0, PmiClient API takes PerfDescriptor(s) and returns PMI data as a CpdCollection object. Each CpdCollection could contain a list of CpdData, which has a CpdValue of the following types:
- CpdLong
- CpdStat
- CpdLoad

Version 4.0 PmiClient APIs are supported in this version, however, there are some changes. The data hierarchy is changed in some PMI modules, notably the enterprise bean module and HTTP sessions module. If you have an existing PmiClient application, and you want to run it against WebSphere Application Server Version 5.0 or above, you might have to update the PerfDescriptor(s) based on the new PMI data hierarchy. Also, the getDataName and getDataId methods in PmiClient are changed to be non-static methods in order to support multiple WebSphere Application Server versions. You might have to update your existing application which uses these two methods.

Example: Performance Monitoring Infrastructure client (Version 4.0)

The following is a list of example Performance Monitoring Infrastructure (PMI) client code from (Version 4.0):

```
/**
 * This is a sample code to show how to use PmiClient to collect PMI data.
 * You will need to use adminconsole to set instrumentation level (a level other
 * than NONE) first.
 *
 * <p>
 * End-to-end code path in 4.0:
 *     PmiTester -> PmiClient -> AdminServer -> appServer
 */


package com.ibm.websphere.pmi;

import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.server.*;
import com.ibm.websphere.pmi.client.*;
import com.ibm.ws.pmi.server.*;
import com.ibm.ws.pmi.perfServer.*;
import com.ibm.ws.pmi.server.modules.*;
import com.ibm.ws.pmi.wire.*;
import java.util.ArrayList;
```

```
/**
 * Sample code to use PmiClient API (old API in 4.0) and get CpdData/CpdCollection objects.
 *
 */
public class PmiTester implements PmiConstants {

    /** a test driver:
     * @param args[0] - node name
     * @param args[1] - port number, optional, default is 2809
     * @param args[2] - connector type, default is RMI
     * @param args[3] - verion (AE, AEs, WAS50), default is WAS50
     *
     */
    public static void main(String[] args) {
        String hostName = null;
        String portNumber = "2809";
        String connectorType = "RMI";
        String version = "WAS50";

        if (args.length < 1) {
            System.out.println("Usage: <host> [<port>] [<connectorType>]
[<version>]");
            return;
        }

        if(args.length >= 1)
            hostName = args[0];
        if(args.length >= 2)
            portNumber = args[1];
        if (args.length >=3)
            connectorType = args[2];
        if (args.length >=4)
            version = args[3];

        try {
            PmiClient pmiClnt = new PmiClient(hostName, portNumber,
version, false, connectorType);

            // uncomment it if you want debug info
            //pmiClnt.setDebug(true);

            // get all the node PerfDescriptor in the domain
            PerfDescriptor[] nodePds = pmiClnt.listNodes();

            if(nodePds == null) {
                System.out.println("no nodes");
                return;
            }

            // get the first node
            String nodeName = nodePds[0].getName();
            System.out.println("after listNodes: " + nodeName);

            //list all the servers on the node
            PerfDescriptor[] serverPds = pmiClnt.listServers(nodePds[0].getName());
            if(serverPds == null || serverPds.length == 0) {
                System.out.println("NO app server in node");
                return;
            }

            // print out all the servers on that node
            for(int j=0; j<serverPds.length; j++) {
                System.out.println("server " + j + ": " + serverPds[j].getName());
            }

            for(int j=0; j<serverPds.length; j++) {
```

```
                System.out.println("server " + j + ": " + serverPds[j].getName());

                // Option: you can call createPerfLevelSpec and then
setInstrumentationLevel to set the level
                // for each server if you want. For example, to set all
the modules to be LEVEL_HIGH for the server j,
                // uncomment the following.
                // PerfLevelSpec[] plds = new PerfLevelSpec[1];
                // plds[0] = pmiClnt.createPerfLevelSpec(null, LEVEL_HIGH);
                // pmiClnt.setInstrumentationLevel(serverPds[j].getNodeName(),
serverPds[j].getServerName(), plds, true);

                // First, list the PerfDescriptor in the server
                PerfDescriptor[] myPds = pmiClnt.listMembers(serverPds[j]);

                // check returned PerfDescriptor
                if(myPds == null) {
                    System.out.println("null from listMembers");
                    continue;
                }

                // you can  add the pds in which you are interested to PerfDescriptorList
                PerfDescriptorList pdList = new PerfDescriptorList();
                for(int i=0; i<myPds.length; i++) {
                    // Option 1: you can recursively call listMembers for each myPds
                    //               and find the one you are interested. You can call
listMembers
                    //               until individual data level and after that level
you will null from listMembers.
                    // e.g., PerfDescriptor[] nextPds = pmiClnt.listMembers(myPds[i]);

                    // Option 2: you can filter these pds  before adding to pdList
                    System.out.println("add to pdList: " + myPds[i].getModuleName());
                    pdList.addDescriptor(myPds[i]);
                    if( i % 2 == 0)
                        pmiClnt.add(myPds[i]);
                }

                // call gets method to get the CpdCollection[] corresponding to pdList
                CpdCollection[] cpdCols = pmiClnt.gets(pdList, true);

                if(cpdCols == null) {
                    // check error
                    if(pmiClnt.getErrorCode() >0)
                        System.out.println(pmiClnt.getErrorMessage());
                    continue;
                }

                for(int i=0; i<cpdCols.length; i++) {
                    // simple print them
                    //System.out.println(cpdCols[i].toString());

                    // Or call processCpdCollection to get each data
                    processCpdCollection(cpdCols[i], "");
                }

                // Or call gets() method to add the CpdCollection[] for whatever
there by calling pmiClnt.add().
                System.out.println("\n\n\n ---- get data using gets(true) ----- ");
                cpdCols = pmiClnt.gets(true);

                if(cpdCols == null) {
                    // check error
                    if(pmiClnt.getErrorCode() >0)
                        System.out.println(pmiClnt.getErrorMessage());
                    continue;
                }
```

```
                for(int i=0; i<cpdCols.length; i++) {
                    // simple print out the whole collection
                    System.out.println(cpdCols[i].toString());

                    // Option: refer processCpdCollection to get each data
                }
            }

        }
        catch(Exception ex) {
            System.out.println("Exception calling CollectorAE");
            ex.printStackTrace();
        }
    }

    /**
     * show the methods to retrieve individual data
     */
    private static void processCpdCollection(CpdCollection cpdCol, String indent) {
        CpdData[] dataList = cpdCol.dataMembers();
        String myindent = indent;

        System.out.println("\n" + myindent + "--- CpdCollection "
+ cpdCol.getDescriptor().getName() + " ---");
        myindent += "   ";
        for(int i=0; i<dataList.length; i++) {
            if (dataList[i] == null)
                continue;

            // if you want to get static info like name, description, etc
            PmiDataInfo dataInfo = dataList[i].getPmiDataInfo();
            // call getName(), getDescription() on dataInfo;

            CpdValue cpdVal = dataList[i].getValue();
            if(cpdVal.getType() == TYPE_STAT) {
                CpdStat cpdStat = (CpdStat)cpdVal;
                double mean = cpdStat.mean();
                double sumSquares = cpdStat.sumSquares();
                int count = cpdStat.count();
                double total = cpdStat.total();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                + " type=stat mean=" + mean);
                // you can print more values like sumSquares, count,etc here
            }
            else if(cpdVal.getType() == TYPE_LOAD) {
                CpdLoad cpdLoad = (CpdLoad)cpdVal;
                long time = cpdLoad.getTime();
                double mean = cpdLoad.mean();
                double currentLevel = cpdLoad.getCurrentLevel();
                double integral = cpdLoad.getIntegral();
                double timeWeight = cpdLoad.getWeight();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                + " type=load mean=" + mean + " currentLevel="
+ currentLevel);
                // you can print more values like sumSquares, count,etc here
            }
            else if(cpdVal.getType() == TYPE_LONG) {
                CpdValue cpdLong = (CpdValue)cpdVal;
                long value = (long)cpdLong.getValue();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                + " type=long value=" + value);
            }
            else if(cpdVal.getType() == TYPE_DOUBLE) {
                CpdValue cpdDouble = (CpdValue)cpdVal;
                double value = cpdDouble.getValue();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
```

```
                                + " type=double value=" + value);
            }
            else if(cpdVal.getType() == TYPE_INT) {
                CpdValue cpdInt = (CpdValue)cpdVal;
                int value = (int)cpdInt.getValue();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                + " type=int value=" + value);
            }
        }

        // recursively go through the subcollection
        CpdCollection[] subCols = cpdCol.subcollections();
        for(int i=0; i<subCols.length; i++) {
            processCpdCollection(subCols[i], myindent);
        }
    }

    /**
     * show the methods to navigate CpdCollection
     */
    private static void report(CpdCollection col) {
        System.out.println("\n\n");
        if(col==null) {
            System.out.println("report: null CpdCollection");
            return;
        }
        System.out.println("report - CpdCollection ");
        printPD(col.getDescriptor());
        CpdData[] dataMembers = col.dataMembers();
        if(dataMembers != null) {
            System.out.println("report CpdCollection: dataMembers is "
+ dataMembers.length);
            for(int i=0; i<dataMembers.length; i++) {
                CpdData data = dataMembers[i];
                printPD(data.getDescriptor());
            }
        }
        CpdCollection[] subCollections = col.subcollections();
        if(subCollections != null) {
            for(int i=0; i<subCollections.length; i++) {
                report(subCollections[i]);
            }
        }
    }

    private static void printPD(PerfDescriptor pd) {
        System.out.println(pd.getFullName());
    }

}
```

Example: Performance Monitoring Infrastructure client with new data structure

This page provides example code using Performance Monitoring Infrastructure (PMI) client with the new data structure.

```
import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.stat.*;
import com.ibm.websphere.pmi.client.*;
import com.ibm.websphere.management.*;
import com.ibm.websphere.management.exception.*;
import java.util.*;
import javax.management.*;
import java.io.*;

/**
 * Sample code to use PmiClient API (new JMX-based API in 5.0) and
get Statistic/Stats objects.
```

```
 */

public class PmiClientTest implements PmiConstants {

    static PmiClient pmiClnt = null;
    static String nodeName = null;
    static String serverName = null;
    static String portNumber = null;
    static String connectorType = null;
    static boolean success = true;


    /**
     * @param args[0] host
     * @param args[1] portNumber, optional, default is 2809
     * @param args[2] connectorType, optional, default is RMI connector
     * @param args[3]serverName, optional, default is the first server found
     */
    public static void main(String[] args) {

        try {

            if(args.length < 1) {
                System.out.println("Parameters: host [portNumber]
[connectorType] [serverName]");
                return;
            }

            // parse arguments and create an instance of PmiClient
            nodeName = args[0];

            if (args.length > 1)
            portNumber = args[1];

            if (args.length > 2)
            connectorType = args[2];

            // create an PmiClient object
            pmiClnt = new PmiClient(nodeName, portNumber, "WAS50", false, connectorType);

            // Uncomment it if you want to debug any problem
            //pmiClnt.setDebug(true);

            // update nodeName to be the real host name
            // get all the node PerfDescriptor in the domain
               PerfDescriptor[] nodePds = pmiClnt.listNodes();
               if(nodePds == null) {
               System.out.println("no nodes");
               return;
                                  }
           // get the first node
              nodeName = nodePds[0].getName();
            System.out.println("use node " + nodeName);

            if (args.length == 4)
                serverName = args[3];
            else { // find the server you want to get PMI data
                // get all servers on this node
                PerfDescriptor[] allservers = pmiClnt.listServers(nodeName);
                if (allservers == null || allservers.length == 0) {
                    System.out.println("No server is found on node " + nodeName);
                    System.exit(1);
                }

                // get the first server on the list. You may want to get a different server
                serverName = allservers[0].getName();
                System.out.println("Choose server " + serverName);
```

```
}

// get all MBeans
ObjectName[] onames = pmiClnt.listMBeans(nodeName, serverName);

// Cache the MBeans we are interested
ObjectName perfOName = null;
ObjectName serverOName = null;
ObjectName wlmOName = null;
ObjectName ejbOName = null;
ObjectName jvmOName = null;
ArrayList myObjectNames = new ArrayList(10);

// get the MBeans we are interested in
if(onames != null) {
    System.out.println("Number of MBeans retrieved= " + onames.length);
    AttributeList al;
    ObjectName on;
    for(int i=0; i<onames.length; i++) {
        on = onames[i];
        String type = on.getKeyProperty("type");

        // make sure PerfMBean is there.
        // Then randomly pick up some MBeans for the test purpose
        if(type != null && type.equals("Server"))
            serverOName = on;
        else if(type != null && type.equals("Perf"))
            perfOName = on;
        else if(type != null && type.equals("WLM")) {
            wlmOName = on;
        }
        else if(type != null && type.equals("EntityBean")) {
            ejbOName = on;

            // add all the EntityBeans to myObjectNames
            myObjectNames.add(ejbOName);  // add to the list
        }
        else if(type != null && type.equals("JVM")) {
            jvmOName = on;
        }
    }

    // set monitoring level for SERVER MBean
    testSetLevel(serverOName);

    // get Stats objects
    testGetStats(myObjectNames);

    // if you know the ObjectName(s)
    testGetStats2(new ObjectName[]{jvmOName, ejbOName});

    // assume you are only interested in a server data in WLM MBean,
    // then you will need to use StatDescriptor and MBeanStatDescriptor
    // Note that wlmModule is only available in ND version
    StatDescriptor sd = new StatDescriptor(new String[] {"wlmModule.server"});
    MBeanStatDescriptor msd = new MBeanStatDescriptor(wlmOName, sd);
    Stats wlmStat = pmiClnt.getStats(nodeName, serverName, msd, false);
    if (wlmStat != null)
        System.out.println("\n\n WLM server data\n\n + " + wlmStat.toString());
    else
        System.out.println("\n\n No WLM server data is availalbe.");

    // how to find all the MBeanStatDescriptors
    testListStatMembers(serverOName);

    // how to use update method
    testUpdate(jvmOName, false, true);
```

```
        }
        else {
            System.out.println("No ObjectNames returned from Query" );
        }

    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        System.out.println("Exception = " +e);
        e.printStackTrace();
        success = false;
    }


    if(success)
        System.out.println("\n\n All tests are passed");
    else
        System.out.println("\n\n Some tests are failed. Check for the exceptions");

}

/**
 * construct an array from the ArrayList
 */
private static MBeanStatDescriptor[] getMBeanStatDescriptor(ArrayList msds) {
    if(msds == null || msds.size() == 0)
        return null;

    MBeanStatDescriptor[] ret = new MBeanStatDescriptor[msds.size()];
    for(int i=0; i<ret.length; i++)
        if(msds.get(i) instanceof ObjectName)
            ret[i] = new MBeanStatDescriptor((ObjectName)msds.get(i));
        else
            ret[i] = (MBeanStatDescriptor)msds.get(i);
    return ret;
}

/**
 * Sample code to navigate and display the data value from the Stats object.
 */
private static void processStats(Stats stat) {
    processStats(stat, "");
}

/**
 * Sample code to navigate and display the data value from the Stats object.
 */
private static void processStats(Stats stat, String indent) {
    if(stat == null)  return;

    System.out.println("\n\n");

    // get name of the Stats
    String name = stat.getName();
    System.out.println(indent + "stats name=" + name);

    // Uncomment the following lines to list all the data names
    /*
    String[] dataNames = stat.getStatisticNames();
    for (int i=0; i<dataNames.length; i++)
        System.out.println(indent + "    " + "data name=" + dataNames[i]);
    System.out.println("\n");
    */

    // list all datas
    com.ibm.websphere.management.statistics.Statistic[] allData = stat.getStatistics();
```

```
        // cast it to be PMI's Statistic type so that we can have get more
        Statistic[] dataMembers = (Statistic[])allData;
        if(dataMembers != null) {
            for(int i=0; i<dataMembers.length; i++)  {
                System.out.print(indent + "    " + "data name="
+ PmiClient.getNLSValue(dataMembers[i].getName())
                                + ", description="
+ PmiClient.getNLSValue(dataMembers[i].getDescription())
                                + ", unit=" + PmiClient.getNLSValue(dataMembers[i].getUnit())
                                + ", startTime=" + dataMembers[i].getStartTime()
                                + ", lastSampleTime=" + dataMembers[i].getLastSampleTime());
                if(dataMembers[i].getDataInfo().getType() == TYPE_LONG) {
                    System.out.println(", count="
+ ((CountStatisticImpl)dataMembers[i]).getCount());
                }
                else if(dataMembers[i].getDataInfo().getType() == TYPE_STAT) {
                    TimeStatisticImpl data = (TimeStatisticImpl)dataMembers[i];
                    System.out.println(", count=" + data.getCount()
                                       + ", total=" + data.getTotal()
                                       + ", mean=" + data.getMean()
                                       + ", min=" + data.getMin()
                                       + ", max=" + data.getMax());
                }
                else if(dataMembers[i].getDataInfo().getType() == TYPE_LOAD) {
                    RangeStatisticImpl data = (RangeStatisticImpl)dataMembers[i];
                    System.out.println(", current=" + data.getCurrent()
                                       + ", lowWaterMark=" + data.getLowWaterMark()
                                       + ", highWaterMark=" + data.getHighWaterMark()
                                       + ", integral=" + data.getIntegral()
                                       + ", avg=" + data.getMean());
                }
            }
        }

        // recursively for sub-stats
        Stats[] substats = (Stats[])stat.getSubStats();
        if(substats == null || substats.length == 0)
            return;
        for(int i=0; i<substats.length; i++) {
            processStats(substats[i], indent + "    ");
        }
    }

    /**
     * test set level and verify using get level
     */
    private static void testSetLevel(ObjectName mbean) {
        System.out.println("\n\n testSetLevel\n\n");
        try {
            // set instrumentation level to be high for the mbean
            MBeanLevelSpec spec = new MBeanLevelSpec(mbean, null, PmiConstants.LEVEL_HIGH);
            pmiClnt.setStatLevel(nodeName, serverName, spec, true);
            System.out.println("after setInstrumentaionLevel high on server MBean\n\n");

            // get all instrumentation levels
            MBeanLevelSpec[] mlss = pmiClnt.getStatLevel(nodeName, serverName, mbean, true);

            if(mlss == null)
                System.out.println("error: null from getInstrumentationLevel");
            else {
                for(int i=0; i<mlss.length; i++)
                    if(mlss[i] != null) {
                        // get the ObjectName, StatDescriptor,
and level out of MBeanStatDescriptor
                        int mylevel = mlss[i].getLevel();
                        ObjectName myMBean = mlss[i].getObjectName();
                        StatDescriptor mysd = mlss[i].getStatDescriptor();  // may be null
```

```
                        // Uncomment it to print all the mlss
                        //System.out.println("mlss " + i + ":, " + mlss[i].toString());
                    }
                }
            }
            catch(Exception ex) {
                new AdminException(ex).printStackTrace();
                ex.printStackTrace();
                System.out.println("Exception in testLevel");
                success = false;
            }
        }

        /**
         * Use listStatMembers method
         */
        private static void testListStatMembers(ObjectName mbean) {

            System.out.println("\n\ntestListStatMembers \n");
            // listStatMembers and getStats
            // From server MBean until the bottom layer.
            try {
                MBeanStatDescriptor[] msds = pmiClnt.listStatMembers(nodeName, serverName, mbean);
                if(msds == null) return;
                System.out.println(" listStatMembers for server MBean, num members
(i.e. top level modules) is " + msds.length);


                for(int i=0; i<msds.length; i++) {
                    if(msds[i] == null)  continue;

                    // get the fields out of MBeanStatDescriptor if you need them
                    ObjectName myMBean = msds[i].getObjectName();
                    StatDescriptor mysd = msds[i].getStatDescriptor();      // may be null

                    // uncomment if you want to print them out
                    //System.out.println(msds[i].toString());
                }

                for(int i=0; i<msds.length; i++) {
                    if(msds[i] == null)  continue;
                    System.out.println("\n\nlistStatMembers for msd=" + msds[i].toString());
                    MBeanStatDescriptor[] msds2 =
pmiClnt.listStatMembers(nodeName, serverName, msds[i]);

                    // you get msds2 at the second layer now and the
listStatMembers can be called recursively
                    // until it returns now.
                }

            }
            catch(Exception ex) {
                new AdminException(ex).printStackTrace();
                ex.printStackTrace();
                System.out.println("Exception in testListStatMembers");
                success = false;
            }

        }

        /**
         * Test getStats method
         */
        private static void testGetStats(ArrayList mbeans) {
            System.out.println("\n\n testgetStats\n\n");
            try {
                Stats[] mystats = pmiClnt.getStats(nodeName,
```

```
serverName, getMBeanStatDescriptor(mbeans), true);

            // navigate each of the Stats object and get/display the value
            for(int k=0; k<mystats.length; k++) {
                processStats(mystats[k]);
            }

        }
        catch(Exception ex) {
            new AdminException(ex).printStackTrace();
            ex.printStackTrace();
            System.out.println("exception from testGetStats");
            success = false;
        }
    }

    /**
     * Test getStats method
     */
    private static void testGetStats2(ObjectName[] mbeans) {
        System.out.println("\n\n testGetStats2\n\n");
        try {
            Stats[] statsArray = pmiClnt.getStats(nodeName, serverName, mbeans, true);

            // You can call toString to simply display all the data
            if(statsArray != null) {
                for(int k=0; k<statsArray.length; k++)
                    System.out.println(statsArray[k].toString());
            }
            else
                System.out.println("null stat");
        }
        catch(Exception ex) {
            new AdminException(ex).printStackTrace();
            ex.printStackTrace();
            System.out.println("exception from testGetStats2");
            success = false;
        }
    }

    /**
     * test update method
     */
    private static void testUpdate(ObjectName oName, boolean keepOld,
boolean recursiveUpdate) {
        System.out.println("\n\n testUpdate\n\n");
        try {
            // set level to be NONE
            MBeanLevelSpec spec = new MBeanLevelSpec(oName, null, PmiConstants.LEVEL_NONE);
            pmiClnt.setStatLevel(nodeName, serverName, spec, true);


            // get data now - one is non-recursive and the other is recursive
            Stats stats1 = pmiClnt.getStats(nodeName, serverName, oName, false);
            Stats stats2 = pmiClnt.getStats(nodeName, serverName, oName, true);

            // set level to be HIGH
            spec = new MBeanLevelSpec(oName, null, PmiConstants.LEVEL_HIGH);
            pmiClnt.setStatLevel(nodeName, serverName, spec, true);

            Stats stats3 = pmiClnt.getStats(nodeName, serverName, oName, true);
            System.out.println("\n\n stats3 is");
            processStats(stats3);

            stats1.update(stats3, keepOld, recursiveUpdate);
            System.out.println("\n\n update stats1");
            processStats(stats1);
```

```
            stats2.update(stats3, keepOld, recursiveUpdate);
            System.out.println("\n\n update stats2");
            processStats(stats2);

        }
        catch(Exception ex) {
            System.out.println("\n\n Exception in testUpdate");
            ex.printStackTrace();
            success = false;
        }

    }


}
```

Performance servlet (PerfServlet)

The PerfServlet is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle.

The servlet provides a way to use an HTTP request to query the performance metrics for an entire WebSphere Application Server administrative domain. Because the servlet provides the performance data through HTTP, issues such as firewalls are trivial to resolve.

The PerfServlet provides the performance data output as an XML document, as described in the provided document type description (DTD). In the XML structure, the leaves of the structure provide the actual observations of performance data and the paths to the leaves that provide the context.

Starting with version 6.0, the PerfServlet in WebSphere Application Server uses the JMX Perf MBean interface to retrieve the PMI data and outputs an XML document that uses the J2EE 1.4 Performance Data Framework to describe the statistics. The PerfSerlvet in can also provide an output that is compatible with the PerfServlet 5.0. To provide PerfServlet 5.0 compatible output it uses the PMI client interface.

The performance servlet .ear file PerfServletApp.ear is located in the WAS_HOME/installableApps directory, where *WAS_HOME* is the installation path for WebSphere Application Server.

The performance servlet is deployed exactly as any other servlet. To use it, follow these steps:

1. Deploy the servlet on a single application server instance within the domain.
2. After the servlet deploys, you can invoke it to retrieve performance data for the entire domain. Invoke the performance servlet by accessing the following default URL:

   `http://hostname/wasPerfTool/servlet/perfservlet`

The performance servlet provides performance data output as an XML document, as described by the provided document type definition (DTD). The DTD is located inside the PerfServletApp.ear file.

PerfServlet input

The PerfServlet input and output is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle

The PerfServlet is deployed in one of the application server instance within the domain. By default, the PerfServlet collects all of the performance data across a WebSphere Application Server cell. However, it is possible to limit the data returned by the servlet to either a specific node, server, or PMI module:

Node .The servlet can limit the information it provides to a specific host by using the node parameter. For example, to limit the data collection to the node ‘rjones’, invoke the following URL:

`http://hostname/wasPerfTool/servlet/perfservlet?node=rjones`

Server

The servlet can limit the information it provides to a specific server by using the server parameter. For example, in order to limit the data collection to the ‘testserver’ server on all nodes, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?server=testserver
```

To limit the data collection to the ‘testserver’ server located on the host ‘rjones’, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?node=rjones&server=testserver
```

Module

The servlet can limit the information it provides to a specific PMI module by using the module parameter. You can request multiple modules by using the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?module=beanModule+jvmRuntimeModule
```

For example, to limit the data collection to the beanModule on all servers and nodes, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?module=beanModule
```

To limit the data collection to the beanModule on the server ‘testserver’ on the node rjones, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?node=rjones&server=testserver&module=beanModule
```

To find the list of the modules, invoke the PerfServlet help with the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?action=help
```

When the performance servlet is first initialized, it retrieves the list of nodes and servers within the domain in which it is deployed. Because the collection of this data is expensive, the performance servlet holds this information as a cached list. If a new node is added to the domain or a new server is started, the performance servlet does not automatically retrieve the information about the newly created element. To force the servlet to refresh its configuration, you must add the refreshConfig parameter to the invocation as follows:

```
http://hostname/wasPerfTool/servlet/perfservlet?refreshConfig=true
```

You may want to configure other parameters of the performance servlets according to your specific needs. You can define the host, port number, connector type, and a user name and password.

- **Host**. This defines the host name where the server is running. The default value is ″localhost.″ For base installations, use ″localhost″ or ″host″ where application server is running. In Network Deployment installations, use the Deployment Manager’s host name.

  Note: There can be more than one server instance in Network Deployment installations and a performance servlet can be installed in any one of the servers.
- **Port**. This is the port through which the server will connect. The default value is ’8880’ (SOAP connector port in base installation). In a base installation, use the application server SOAP or RMI connector port. In a Network Deployment installation, use the Deployment Manager’s SOAP or RMI port.

  Note: The port numbers for SOAP/RMI connector can be configured in the administrative console under **Servers** > **Application Servers** > *server_name* > **End Points**.
- **Connector**. The connector type can be either SOAP or RMI. The default value is SOAP.
- **Username**. If security is enabled provide the user name.
- **Password**. If security is enabled provide the password.

PerfServlet output

The PerfServlet input and output is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle.

The performance servlet .ear file PerfServletApp.ear is located in the `WAS_HOME/installableApps` directory.

The PerfServlet 6.0 provides output using the J2EE 1.4 Performance Data Framework. By default, the PerfServlet output is in 6.0 format. PerfServlet can provide the output in 5.0 format using the version parameter:

```
http://hostname/wasPerfTool/servlet/perfservlet?version=5
```

Refer to “PMI data classification” on page 2106 for details about the Performance Data Framework.

PerfServlet 5.0 output details: The following section describes the PerfServlet 5.0 output. There are three types of leaves or output formats within the XML structure: PerfNumericInfo, PerfStatInfo, and PerfLoadInfo.

PerfNumericInfo:

When each invocation of the performance servlet retrieves the performance values from Performance Monitoring Infrastructure (PMI), some of the values are raw counters that record the number of times a specific event occurs during the lifetime of the server. If a performance observation is of the type PerfNumericInfo, the value represents the raw count of the number of times this event has occurred since the server started. This information is important to note because the analysis of a single document of data provided by the performance servlet might not be useful for determining the current load on the system. To determine the load during a specific interval of time, it might be necessary to apply simple statistical formulas to the data in two or more documents provided during this interval.

The PerfNumericInfo type has the following attributes:

time Specifies the time when the observation was collected (Java System.currentTimeMillis)

uid Specifies the PMI identifier for the observation

val Specifies the raw counter value

The following document fragment represents the number of loaded servlets. The path providing the context of the observation is not shown:

```
<numLoadedServlets>
     <PerfNumericData time="988162913175" uid="pmi1" val="132"/>
</numLoadedServlets>
```

PerfStatInfo:

When each invocation of the performance servlet retrieves the performance values from PMI, some of the values are stored as statistical data. Statistical data records the number of occurrences of a specific event, as the PerfNumericInfo type does. In addition, this type has sum of squares, mean, and total for each observation. This value is relative to when the server started.

The PerfStatInfo type has the following attributes:

time Specifies the time when the observation was collected (Java System.currentTimeMillis)

uid Specifies the PMI identifier for the observation

num Specifies the number of observations

sum_of_squares
Specifies the sum of the squares of the observations

total Specifies the sum of the observations

mean Specifies the mean (total number) for this counter

The following fragment represents the response time of an object. The path providing the context of the observation is not shown:

```
<responseTime>
     <PerfStatInfo mean="1211.5" num="5" sum_of_squares="3256265.0"
                   time="9917644193057" total="2423.0" uid="pmi13"/>
</responseTime>
```

PerfLoadInfo:

When each invocation of the performance servlet retrieves the performance values from PMI, some of the values are stored as a load. Loads record values as a function of time; they are averages. This value is relative to when the server started.

The PerfLoadInfo type has the following attributes:

time Specifies the time when the observation was collected (Java System.currentTimeMillis)

uid Specifies the PMI identifier for the observation

currentValue
Specifies the current value for this counter

integral
Specifies the time-weighted sum

timeSinceCreate
Specifies the elapsed time in milliseconds since this data was created in the server

mean Specifies time-weighted mean (integral/timeSinceCreate) for this counter

The following fragment represents the number of concurrent requests. The path providing the context of the observation is not shown:

```
<poolSize>
     <PerfLoadInfo currentValue="1.0" integral="534899.0" mean="0.9985028962051592"
                   time="991764193057" timeSinceCreate="535701.0" uid="pmi5"/>
</poolSize>
```

Related concepts

“PerfServlet input” on page 2236
The PerfServlet input and output is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle

Related tasks

“Performance servlet (PerfServlet)” on page 2236
The PerfServlet is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle.

Using the JMX interface to develop your own monitoring application

You can use AdminClient API to get Performance Monitoring Infrastructure (PMI) data by using either PerfMBean or individual MBeans.

You can invoke methods on MBeans through the AdminClient Java Management Extension (JMX) interface. You can use AdminClient API to get Performance Monitoring Infrastructure (PMI) data by using either PerfMBean or individual MBeans. See information about using individual MBeans at bottom of this article.

Individual MBeans provide the Stats attribute from which you can get PMI data. The PerfMBean provides extended methods for PMI administration and more efficient ways to access PMI data. To set the PMI module instrumentation level, you must invoke methods on PerfMBean. To query PMI data from multiple MBeans, it is faster to invoke the getStatsArray method in PerfMBean than to get the Stats attribute from multiple individual MBeans. Perf MBean can provide PMI data from multiple MBeans using a single JMX call, but multiple JMX calls have to be made through individual MBeans.

See the topic ″Developing an administrative client program″ for more information on AdminClient JMX.

After the performance monitoring service is enabled and the application server is started or restarted, a PerfMBean is located in each application server giving access to PMI data. To use PerfMBean:

1. Create an instance of AdminClient. When using AdminClient API, you need to first create an instance of AdminClient by passing the host name, port number and connector type.

   The example code is:

   ```
   AdminClient ac = null;
   java.util.Properties props = new java.util.Properties();
   props.put(AdminClient.CONNECTOR_TYPE, connector);
   props.put(AdminClient.CONNECTOR_HOST, host);
   props.put(AdminClient.CONNECTOR_PORT, port);
   try {
       ac = AdminClientFactory.createAdminClient(props);
   }
   catch(Exception ex) {
       failed = true;
       new AdminException(ex).printStackTrace();
       System.out.println("getAdminClient: exception");
   }
   ```

2. Use AdminClient to query the MBean ObjectNames Once you get the AdminClient instance, you can call queryNames to get a list of MBean ObjectNames depending on your query string. To get all the ObjectNames, you can use the following example code. If you have a specified query string, you will get a subset of ObjectNames.

   ```
   javax.management.ObjectName on = new javax.management.ObjectName("WebSphere:*");
         Set objectNameSet= ac.queryNames(on, null);
   // you can check properties like type, name, and process to find a specified ObjectName
   ```

   After you get all the ObjectNames, you can use the following example code to get all the node names:

   ```
   HashSet nodeSet = new HashSet();
         for(Iterator i = objectNameSet.iterator(); i.hasNext(); on =
   (ObjectName)i.next()) {
               String type = on.getKeyProperty("type");
               if(type != null && type.equals("Server")) {
   nodeSet.add(servers[i].getKeyProperty("node"));
               }
   }
   ```

   Note, this will only return nodes that are started. To list running servers on the node, you can either check the node name and type for all the ObjectNames or use the following example code:

   ```
       StringBuffer oNameQuery= new StringBuffer(41);
          oNameQuery.append("WebSphere:*");
          oNameQuery.append(",type=").append("Server");
          oNameQuery.append(",node=").append(mynode);

          oSet= ac.queryNames(new ObjectName(oNameQuery.toString()), null);
          Iterator i = objectNameSet.iterator ();
     while (i.hasNext ()) {
     on=(objectName) i.next();
    String process= on[i].getKeyProperty("process");
    serversArrayList.add(process);
   }
   ```

3. Get the PerfMBean ObjectName for the application server from which you want to get PMI data. Use this example code:

```
  for(Iterator i = objectNameSet.iterator(); i.hasNext(); on = (ObjectName)i.next()) {
     // First make sure the node name and server name is what you want
     // Second, check if the type is Perf
            String type = on.getKeyProperty("type");
     String node = on.getKeyProperty("node");
     String process= on.getKeyProperty("process");
            if (type.equals("Perf") && node.equals(mynode) &
& server.equals(myserver)) {
  perfONname = on;
            }
 }
```

4. Invoke operations on PerfMBean through the AdminClient. Once you get the PerfMBean(s) in the application server from which you want to get PMI data, you can invoke the following operations on the PerfMBean through AdminClient API:

```
- setStatisticSet: Enable PMI data using the pre-defined statistic sets.
 params[0] = new String[] { com.ibm.websphere.pmi.stat.StatConstants.STATISTIC_SET_EXTENDED};
 signature = new String[] {"java.lang.String"};
 ac.invoke (perfONname, "setStatisticSet", params, signature);

- getStatisticSet: Returns the current statistic set.
 ac.invoke (perfONname, "getStatisticSet", null, null);

- setCustomSetString: Customizing PMI data that is enabled using fine-grained control.
This method allows to enable or disable statistics selectively. The format of the custom
set specification string is STATS_NAME=ID1,ID2,ID3 seperated by ':', where STATS_NAME
and IDs are defined in WS*Stat interfaces in com.ibm.websphere.pmi.stat package.
 params[0] = new String (WSJVMStats.NAME + "=" + WSJVMStats.HeapSize);
 params[1] = new Boolean (false);
 signature = new String[] {"java.lang.String", "java.lang.Boolean"};
 ac.invoke (perfONname, "setCustomSetString", null, null);

- getCustomSetString: Returns the current custom set specification as a string
 ac.invoke (perfONname, "getCustomSetString", null, null);

- setInstrumentationLevel: set the instrumentation level
            params[0] = new MBeanLevelSpec(objectName, new int[]{WSJVMStats.HEAPSIZE});
            params[1] = new Boolean(true);
            signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanLevelSpec",
"java.lang.Boolean"};
            ac.invoke(perfONname, "setInstrumentationLevel", params, signature);

- getInstrumentationLevel: get the instrumentation level
            params[0] = objectName;
            params[1] = new Boolean(recursive);
            String[] signature= new String[]{
"javax.management.ObjectName", "java.lang.Boolean"};
            MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfONname,
"getInstrumentationLevel", params, signature);

- setInstrumentationLevel: set the instrumentation level (deprecated in V6.0)
            params[0] = new MBeanLevelSpec(objectName, optionalSD, level);
            params[1] = new Boolean(true);
            signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanLevelSpec",
"java.lang.Boolean"};
            ac.invoke(perfONname, "setInstrumentationLevel", params, signature);

- getInstrumentationLevel: get the instrumentation level (deprecated in V6.0)
            Object[] params = new Object[2];
            params[0] = new MBeanStatDescriptor(objectName, optionalSD);
            params[1] = new Boolean(recursive);
            String[] signature= new String[]{
"com.ibm.websphere.pmi.stat.MBeanStatDescriptor", "java.lang.Boolean"};
            MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfONname,
```

```
"getInstrumentationLevel", params, signature);

- getConfigs: get PMI static config info for all the MBeans
            configs = (PmiModuleConfig[])ac.invoke(perfONname, "getConfigs", null, null);

- getConfig: get PMI static config info for a specific MBean
     ObjectName[] params = {objectName};
     String[] signature= { "javax.management.ObjectName" };
            config = (PmiModuleConfig)ac.invoke(perfONname, "getConfig", params,
signature);

- getStatsObject: you can use either ObjectName or MBeanStatDescriptor
            Object[] params    = new Object[2];
            params[0] = objectName;  // either ObjectName or or MBeanStatDescriptor
            params[1] = new Boolean(recursive);
            String[] signature = new String[] { "javax.management.ObjectName",
"java.lang.Boolean"};
            Stats stats  = (Stats)ac.invoke(perfONname, "getStatsObject", params,
signature);

  Note: The returned data only have dynamic information (value and time stamp).
See PmiJmxTest.java for additional code to link the configuration information with the
returned data.

- getStatsArray: you can use either ObjectName or MBeanStatDescriptor
            ObjectName[] onames = new ObjectName[]{objectName1, objectName2};
            Object[] params = new Object[]{onames, new Boolean(true)};
            String[] signature = new String[]{"[Ljavax.management.ObjectName;",
"java.lang.Boolean"};
            Stats[] statsArray = (Stats[])ac.invoke(perfONname, "getStatsArray",
params, signature);

  Note: The returned data only have dynamic information (value and time stamp).
See PmiJmxTest.java for additional code to link the configuration information with the
returned data.

- listStatMembers: navigate the PMI module trees

            Object[] params = new Object[]{mName};
            String[] signature= new String[]{"javax.management.ObjectName"};
            MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke(perfONname,
"listStatMembers", params, signature);

or,

            Object[] params = new Object[]{mbeanSD};
            String[] signature= new String[]
{"com.ibm.websphere.pmi.stat.MBeanStatDescriptor"};
            MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke
(perfONname, "listStatMembers", params, signature);

Refer the API documentation for deprecated classes
```

- **To use an individual MBean:** You need to get the AdminClient instance and the ObjectName for the individual MBean. Then you can simply get the Stats attribute on the MBean.

Example: Administering Java Management Extension-based interface

The page provides example code directly using Java Management Extension (JMX) API. For information on compiling your source code, see ″Compiling your monitoring applications.″

```
package com.ibm.websphere.pmi;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;
import com.ibm.websphere.management.exception.InvalidAdminClientTypeException;
import com.ibm.websphere.management.exception.*;
```

```
import java.util.*;
import javax.management.*;
import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.client.*;
import com.ibm.websphere.pmi.stat.*;

/**
 * Sample code using AdminClient API to get PMI data from PerfMBean
 * and individual MBeans.
 *
 * @ibm-api
 */

public class PmiJmxTest implements PmiConstants
{

    private AdminClient    ac = null;
    private ObjectName     perfOName   = null;
    private ObjectName     serverOName = null;
    private ObjectName     wlmOName    = null;
    private ObjectName     jvmOName    = null;
    private ObjectName     orbtpOName    = null;
    private boolean failed = false;
    private PmiModuleConfig[] configs = null;

    /**
     *  Creates a new test object
     *  (Need a default constructor for the testing framework)
     */
    public PmiJmxTest()
    {
    }

    /**
     * @param args[0] host
     * @param args[1] port, optional, default is 8880
     * @param args[2] connectorType, optional, default is SOAP connector
     *
     */
    public static void main(String[] args)
    {
        PmiJmxTest instance = new PmiJmxTest();

        // parse arguments and create AdminClient object
        instance.init(args);

        // navigate all the MBean ObjectNames and cache those we are interested
        instance.getObjectNames();

        boolean v6 = !(new Boolean(System.getProperty ("websphereV5Statistics"))).
     booleanValue();

        if( v6 )
        {
            // test V6 APIs
            instance.doTestV6();
        }
        else
        {

            // set level, get data, display data
            instance.doTest();

            // test for EJB data
            instance.testEJB();
```

```
        // how to use JSR77 getStats method for individual MBean other than PerfMBean
        instance.testJSR77Stats();
    }

}

/**
 * parse args and getAdminClient
 */
public void init(String[] args)
{

    try
    {
        String  host    = null;
        String  port    = "8880";
        String  connector = AdminClient.CONNECTOR_TYPE_SOAP;
        if(args.length < 1) {
            System.err.println("ERROR: Usage: PmiJmxTest <host> [<port>] [<connector>]");
            System.exit(2);
        }
        else
        {
            host = args[0];

            if (args.length > 1)
                port = args[1];

            if (args.length > 2)
                connector = args[2];
        }

        if(host == null) {
            host = "localhost";
        }
        if(port == null) {
            port = "2809";
        }
        if (connector == null) {
            connector = AdminClient.CONNECTOR_TYPE_SOAP;
        }
        System.out.println("host=" + host + " , port=" + port + ",connector=" + connector);

        //-------------------------------------------------------------------
        // Get the ac object for the AppServer
        //-------------------------------------------------------------------
        System.out.println("main: create the adminclient");
        ac = getAdminClient(host, port, connector);

    }
    catch (Exception ex)
    {
        failed = true;
        new AdminException(ex).printStackTrace();
        ex.printStackTrace();
    }
}

/**
 * get AdminClient using the given host, port, and connector
 */
public AdminClient getAdminClient(String hostStr, String portStr, String
connector) {
    System.out.println("getAdminClient: host=" + hostStr + " , portStr=" + portStr);
    AdminClient ac = null;
    java.util.Properties props = new java.util.Properties();
    props.put(AdminClient.CONNECTOR_TYPE, connector);
```

```
      props.put(AdminClient.CONNECTOR_HOST, hostStr);
      props.put(AdminClient.CONNECTOR_PORT, portStr);

      /* set the following properties if security is enabled and using SOAP
 connector */
      /* The following shows how to set properties for SOAP connector when
 security is enabled.
         See AdminClient javadoc for more info.
      Properties props = new Properties();
     props.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
     props.setProperty(AdminClient.CONNECTOR_PORT, "8880");
     props.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_
 TYPE_SOAP);
     props.setProperty(AdminClient.CONNECTOR_SECURITY_ENABLED, "true");
     props.setProperty(AdminClient.USERNAME, "test2");
    props.setProperty(AdminClient.PASSWORD, "user24test");
      props.setProperty("javax.net.ssl.trustStore",
"C:/WebSphere/AppServer/etc/DummyClientTrustFile.jks");
  props.setProperty("javax.net.ssl.keyStore",
"C:/WebSphere/AppServer/etc/DummyClientKeyFile.jks");
  props.setProperty("javax.net.ssl.trustStorePassword", "WebAS");
  props.setProperty("javax.net.ssl.keyStorePassword", "WebAS");
      */

      try {
          ac = AdminClientFactory.createAdminClient(props);
      }
      catch(Exception ex) {
          failed = true;
          new AdminException(ex).printStackTrace();
          System.out.println("getAdminClient: exception");
      }
      return ac;
  }


  /**
   * get all the ObjectNames.
   */
  public void getObjectNames() {

      try {

          //------------------------------------------------------------------
          // Get a list of object names
          //------------------------------------------------------------------
          javax.management.ObjectName on = new javax.management.ObjectName
 ("WebSphere:*");

          //------------------------------------------------------------------
          // get all objectnames for this server
          //------------------------------------------------------------------
         Set objectNameSet= ac.queryNames(on, null);

          //------------------------------------------------------------------
          // get the object names that we care about: Perf, Server, JVM, WLM (only applicable in ND)
          //------------------------------------------------------------------
          if(objectNameSet != null) {
              Iterator i = objectNameSet.iterator();
              while (i.hasNext()) {
                  on = (ObjectName)i.next();
                  String type = on.getKeyProperty("type");

                  // uncomment it if you want to print the ObjectName for each MBean
                  // System.out.println("\n\n" + on.toString());

                  // find the MBeans we are interested
```

```
                if(type != null && type.equals("Perf")) {
                    System.out.println("\nMBean: perf =" + on.toString());
                    perfOName = on;
                }
                if(type != null && type.equals("Server")) {
                    System.out.println("\nMBean: Server =" + on.toString());
                    serverOName = on;
                }
                if(type != null && type.equals("JVM")) {
                    System.out.println("\nMBean: jvm =" + on.toString());
                    jvmOName = on;
                }
                if(type != null && type.equals("WLMAppServer")) {
                    System.out.println("\nmain: WLM =" + on.toString());
                    wlmOName = on;
                }
                if(type != null && type.equals("ThreadPool"))
                {
                    String name = on.getKeyProperty("name");
                    if (name.equals("ORB.thread.pool"))
                        System.out.println("\nMBean: ORB ThreadPool =" + on.toString());
                    orbtpOName = on;
                }
            }
        }
        else {
            System.err.println("main: ERROR: no object names found");
            System.exit(2);
        }

        // You must have Perf MBean in order to get PMI data.
        if (perfOName == null)
        {
            System.err.println("main: cannot get PerfMBean. Make sure PMI is enabled");
            System.exit(3);
        }
    }
    catch(Exception ex)
    {
        failed = true;
        new AdminException(ex).printStackTrace();
        ex.printStackTrace();
    }

}

/** Test V6 APIs */
public void doTestV6 ()
{
    System.out.println ("\ndoTestV6() output:\n");

    // the following methods are specific to V6 and demonstrates the V6 API..so set the flag to false
    String v5PropFlag = System.setProperty ("websphereV5Statistics", "false");
    try
    {
        Object[] params;
        String[] signature;

        // get current statistic set that is used for monitoring
        System.out.println ("\nCurrent statistic set: " + ac.invoke(perfOName, "getStatisticSet", null, null));


        // get all statistics from the server using Perf MBean
        System.out.println ("\nGet all statistics in PMI tree");
        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatDescriptor;","java.lang.Boolean"};
        params = new Object[] {new StatDescriptor[]{new StatDescriptor(null)}, new Boolean(true)};
```

```
        com.ibm.websphere.pmi.stat.WSStats[] wsStats = (com.ibm.websphere.pmi.stat.WSStats[])
        ac.invoke(perfOName, "getStatsArray", params, signature);
        System.out.println (wsStats[0].toString());


        // get statistics from one JVM MBean using J2EE JMX
        System.out.println ("\nGet JVM statistics using JVM MBean");
        javax.management.j2ee.statistics.Stats j2eeStats = (javax.management.j2ee.statistics.Stats)
        ac.getAttribute(jvmOName, "stats");
        System.out.println (j2eeStats.toString());


        // get statistics from a specific thread pool -- WebContainer thread pool
        System.out.println ("\nGet statistics for a specific thread pool");
        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatDescriptor;","java.lang.Boolean"};

        StatDescriptor webContainerPoolSD = new StatDescriptor (new String[] {WSThreadPoolStats.NAME,
  "WebContainer"});
        params = new Object[] {new StatDescriptor[]{webContainerPoolSD}, new Boolean(true)};

        wsStats = (com.ibm.websphere.pmi.stat.WSStats[])
        ac.invoke(perfOName, "getStatsArray", params, signature);
        System.out.println (wsStats[0].toString());


        // set monitoring to statistic set "extended"
        System.out.println ("\nSet monitoring to statistic set 'Extended'");
        signature = new String[]{"java.lang.String"};
        params = new Object[] {StatConstants.STATISTIC_SET_EXTENDED};

        ac.invoke(perfOName, "setStatisticSet", params, signature);

        // get current statistic set that is used for monitoring
        System.out.println ("\nCurrent statistic set: "+ ac.invoke(perfOName, "getStatisticSet", null, null));


        // selectively enable statistics for all thread pools
        System.out.println ("\nSelectively enable statistics (ActiveCount and PoolSize statistics)
   for thread pool -- fine grained control");

        StatDescriptor threadPoolSD = new StatDescriptor (new String[]
{WSThreadPoolStats.NAME});
            // create a spec object to enable ActiveCount and PoolSize on the thread pool
        StatLevelSpec[] spec = new StatLevelSpec[1];
        spec[0] = new StatLevelSpec (threadPoolSD.getPath(), new int[]
{WSThreadPoolStats.ActiveCount, WSThreadPoolStats.PoolSize});

        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatLevelSpec;","java.lang.Boolean"};
        params = new Object[] {spec, new Boolean(true)};

        ac.invoke(perfOName, "setInstrumentationLevel", params, signature);


        // get current statistic set that is used for monitoring
        System.out.println ("\nCurrent statistic set: "+ ac.invoke(perfOName, "getStatisticSet", null, null));

        // get statistics from all thread pools
        System.out.println ("\nGet statistics from all thread pools");
        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatDescriptor;","java.lang.Boolean"};
        params = new Object[] {new StatDescriptor[]{threadPoolSD},new Boolean(true)};

        wsStats = (com.ibm.websphere.pmi.stat.WSStats[])
        ac.invoke(perfOName, "getStatsArray", params, signature);
        System.out.println (wsStats[0].toString());

    }
    catch (Exception e)
```

```
    {
        e.printStackTrace();
    }

    // set the property to original value
    System.setProperty ("websphereV5Statistics", v5PropFlag);
}

/**
 * Some sample code to set level, get data, and display data. (V5)
 * @deprecated Use 6.0 APIs.
 */
public void doTest()
{
    try
    {
        // first get all the configs  - used to set static info for Stats
        // Note: server only returns the value and time info.
        //       No description, unit, etc is returned with PMI data to reduce communication cost.
        //       You have to call setConfig to bind the static info and Stats data later.
        configs = (PmiModuleConfig[])ac.invoke(perfOName, "getConfigs", null, null);

        // print out all the PMI modules and matching mbean types
        for (int i=0; i<configs.length; i++)
            System.out.println("config: moduleName=" + configs[i].getShortName() +
   ", mbeanType=" + configs[i].getMbeanType());

        // set the instrumentation level for the server
        setInstrumentationLevel(serverOName, null, PmiConstants.LEVEL_HIGH);

        // example to use StatDescriptor.
        // Note WLM module is only available in ND.
        StatDescriptor sd = new StatDescriptor(new String[] {"wlmModule.server"});
        setInstrumentationLevel(wlmOName, sd, PmiConstants.LEVEL_HIGH);

        // example to getInstrumentationLevel
        MBeanLevelSpec[] mlss = getInstrumentationLevel(wlmOName, sd, true);
        // you can call getLevel(), getObjectName(), getStatDescriptor() on mlss[i]

        // get data for the server
        Stats stats = getStatsObject(serverOName, true);
        System.out.println(stats.toString());

        // get data for WLM server submodule
        stats = getStatsObject(wlmOName, sd, true);
        if (stats == null)
            System.out.println("Cannot get Stats for WLM data");
        else
            System.out.println(stats.toString());

        // get data for JVM MBean
        stats = getStatsObject(jvmOName, true);
        processStats(stats);

        // get data for multiple MBeans
        ObjectName[] onames = new ObjectName[]{orbtpOName, jvmOName};
        Object[] params = new Object[]{onames, new Boolean(true)};
        String[] signature = new String[]{"[Ljavax.management.ObjectName;",
  "java.lang.Boolean"};
        Stats[] statsArray = (Stats[])ac.invoke(perfOName, "getStatsArray",
  params, signature);
        // you can call toString or processStats on statsArray[i]

        if (!failed)
            System.out.println("All tests passed");
        else
            System.out.println("Some tests failed");
```

```
    }
    catch(Exception ex)
    {
        new AdminException(ex).printStackTrace();
        ex.printStackTrace();
    }
}


/**
 * Sample code to get level
 */
protected MBeanLevelSpec[] getInstrumentationLevel(ObjectName on, StatDescriptor sd, boolean recursive)
{
    if (sd == null)
        return getInstrumentationLevel(on, recursive);
    System.out.println("\ntest getInstrumentationLevel\n");
    try {
        Object[] params = new Object[2];
        params[0] = new MBeanStatDescriptor(on, sd);
        params[1] = new Boolean(recursive);
        String[] signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanStatDescriptor", "java.lang.Boolean"};
        MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfOName, "getInstrumentationLevel", params, signature)
        return mlss;
    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        System.out.println("getInstrumentationLevel: Exception Thrown");
        return null;
    }
}

/**
 * Sample code to get level
 */
protected MBeanLevelSpec[] getInstrumentationLevel(ObjectName on,
 boolean recursive) {
    if (on == null)
        return null;
    System.out.println("\ntest getInstrumentationLevel\n");
    try {
        Object[] params = new Object[]{on, new Boolean(recursive)};
        String[] signature= new String[]{ "javax.management.ObjectName",
 "java.lang.Boolean"};
        MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfOName,
 "getInstrumentationLevel", params, signature);
        return mlss;
    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        failed = true;
        System.out.println("getInstrumentationLevel: Exception Thrown");
        return null;
    }
}

/**
 * Sample code to set level
 * @deprecated Use 6.0 APIs.
 */
protected void setInstrumentationLevel(ObjectName on, StatDescriptor sd,
 int level) {
    System.out.println("\ntest setInstrumentationLevel\n");
    try {
        Object[] params       = new Object[2];
        String[] signature    = null;
        MBeanLevelSpec[] mlss = null;
```

```
        params[0] = new MBeanLevelSpec(on, sd, level);
        params[1] = new Boolean(true);

        signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanLevelSpec","java.lang.Boolean"};
        ac.invoke(perfOName, "setInstrumentationLevel", params, signature);
    }
    catch(Exception e) {
        failed = true;
        new AdminException(e).printStackTrace();
        System.out.println("setInstrumentationLevel: FAILED: Exception Thrown");
    }
}

/**
 * Sample code to get a Stats object
 * @deprecated Use 6.0 APIs.
 */
public Stats getStatsObject(ObjectName on, StatDescriptor sd, boolean
 recursive) {

    if (sd == null)
        return getStatsObject(on, recursive);

    System.out.println("\ntest getStatsObject\n");
    try {
        Object[] params    = new Object[2];
        params[0] = new MBeanStatDescriptor(on, sd);  // construct MBeanStatDescriptor
        params[1] = new Boolean(recursive);
        String[] signature = new String[] { "com.ibm.websphere.pmi.stat.MBeanStatDescriptor", "java.lang.Boolean"};
        Stats stats  = (Stats)ac.invoke(perfOName, "getStatsObject", params, signature);

        if (stats == null) return null;


        // find the PmiModuleConfig and bind it with the data
        String type = on.getKeyProperty("type");
        if (type.equals(MBeanTypeList.SERVER_MBEAN))
            setServerConfig(stats);
        else
            stats.setConfig(PmiClient.findConfig(configs, on));

        return stats;

    } catch(Exception e) {
        failed = true;
        new AdminException(e).printStackTrace();
        System.out.println("getStatsObject: Exception Thrown");
        return null;
    }
}

/**
 * Sample code to get a Stats object
 */
public Stats getStatsObject(ObjectName on, boolean recursive) {
    if (on == null)
        return null;

    System.out.println("\ntest getStatsObject\n");

    try {
        Object[] params  = new Object[]{on, new Boolean(recursive)};
        String[] signature = new String[] { "javax.management.ObjectName",
 "java.lang.Boolean"};
        Stats stats  = (Stats)ac.invoke(perfOName, "getStatsObject", params,
 signature);
```

```
            // find the PmiModuleConfig and bind it with the data
            String type = on.getKeyProperty("type");
            if (type.equals(MBeanTypeList.SERVER_MBEAN))
                setServerConfig(stats);
            else
                stats.setConfig(PmiClient.findConfig(configs, on));

            return stats;

        }
        catch(Exception e) {
            failed = true;
            new AdminException(e).printStackTrace();
            System.out.println("getStatsObject: Exception Thrown");
            return null;
        }
    }

    /**
     * Sample code to navigate and get the data value from the Stats object.
     */
    private void processStats(Stats stat) {
        processStats(stat, "");
    }

    /**
     * Sample code to navigate and get the data value from the Stats and
Statistic object.
     * @deprecated Use 6.0 APIs.
     */
    private void processStats(Stats stat, String indent) {
        if(stat == null)  return;

        System.out.println("\n\n");

        // get name of the Stats
        String name = stat.getName();
        System.out.println(indent + "stats name=" + name);

        // list data names
        String[] dataNames = stat.getStatisticNames();
        for (int i=0; i<dataNames.length; i++)
            System.out.println(indent + "    " + "data name=" + dataNames[i]);
        System.out.println("");

        // list all datas
        //com.ibm.websphere.management.statistics.Statistic[] allData =
stat.getStatistics();

        // cast it to be PMI's Statistic type so that we can have get more
        // Also show how to do translation.
        //Statistic[] dataMembers = (Statistic[])allData;
        Statistic[] dataMembers = stat.listStatistics();
        if(dataMembers != null) {
            for(int i=0; i<dataMembers.length; i++)  {
                System.out.print(indent + "    " + "data name=" +
PmiClient.getNLSValue(dataMembers[i].getName())
                                + ", description=" + PmiClient.getNLSValue
(dataMembers[i].getDescription())
                                + ", startTime=" + dataMembers[i].getStartTime()
                                + ", lastSampleTime=" + dataMembers[i].getLastSampleTime());
                if(dataMembers[i].getDataInfo().getType() == TYPE_LONG) {
                    System.out.println(", count=" + ((CountStatisticImpl)dataMembers[i]).getCount());
                }
                else if(dataMembers[i].getDataInfo().getType() == TYPE_STAT) {
                    TimeStatisticImpl data = (TimeStatisticImpl)dataMembers[i];
```

```
                System.out.println(", count=" + data.getCount()
                                   + ", total=" + data.getTotal()
                                   + ", mean=" + data.getMean()
                                   + ", min=" + data.getMin()
                                   + ", max=" + data.getMax());
            }
            else if(dataMembers[i].getDataInfo().getType() == TYPE_LOAD) {
                RangeStatisticImpl data = (RangeStatisticImpl)dataMembers[i];
                System.out.println(", current=" + data.getCurrent()
                                   + ", integral=" + data.getIntegral()
                                   + ", avg=" + data.getMean()
                                   + ", lowWaterMark=" + data.getLowWaterMark()
                                   + ", highWaterMark=" + data.getHighWaterMark());
            }
        }
    }

    // recursively for sub-stats
    Stats[] substats = (Stats[])stat.getSubStats();
    if(substats == null || substats.length == 0)
        return;
    for(int i=0; i<substats.length; i++) {
        processStats(substats[i], indent + "    ");
    }
}

/**
 * The Stats object returned from server does not have static config info.
You have to set it on client side.
 */
public void setServerConfig(Stats stats) {
    if(stats == null) return;
    if(stats.getType() != TYPE_SERVER) return;

    PmiModuleConfig config = null;

    Stats[] statList = stats.getSubStats();
    if (statList == null || statList.length == 0)
        return;
    Stats oneStat = null;
    for(int i=0; i<statList.length; i++) {
        oneStat = statList[i];
        if (oneStat == null) continue;
        config = PmiClient.findConfig(configs, oneStat.getStatsType());
 //getName
        if(config != null)
            oneStat.setConfig(config);
        else
        {
            config = getStatsConfig (oneStat.getStatsType());
            if (config != null)
                oneStat.setConfig(config);
            else
                System.out.println("Error: get null config for " + oneStat.getStatsType());
        }
    }
}

/**
 * sample code to show how to get a specific MBeanStatDescriptor
 * @deprecated Use 6.0 APIs.
 */
public MBeanStatDescriptor getStatDescriptor(ObjectName oName, String name) {
    try {
        Object[] params = new Object[]{serverOName};
        String[] signature= new String[]{"javax.management.ObjectName"};
        MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke
```

```
(perfOName, "listStatMembers", params, signature);
         if (msds == null)
             return null;
         for (int i=0; i<msds.length; i++) {
             if (msds[i].getName().equals(name))
                 return msds[i];
         }
         return null;
     }
     catch(Exception e) {
         new AdminException(e).printStackTrace();
         System.out.println("listStatMembers: Exception Thrown");
         return null;
     }

 }

 /**
  * sample code to show you how to navigate MBeanStatDescriptor via
 listStatMembers
  * @deprecated Use 6.0 APIs.
  */
 public MBeanStatDescriptor[] listStatMembers(ObjectName mName) {
     if (mName == null)
         return null;

     try {
         Object[] params = new Object[]{mName};
         String[] signature= new String[]{"javax.management.ObjectName"};
         MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke
  (perfOName, "listStatMembers", params, signature);
         if (msds == null)
             return null;
         for (int i=0; i<msds.length; i++) {
             MBeanStatDescriptor[] msds2 = listStatMembers(msds[i]);
         }
         return null;
     }
     catch(Exception e) {
         new AdminException(e).printStackTrace();
         System.out.println("listStatMembers: Exception Thrown");
         return null;
     }

 }


 /**
  * Sample code to get MBeanStatDescriptors
  * @deprecated Use 6.0 APIs.
  */
 public MBeanStatDescriptor[] listStatMembers(MBeanStatDescriptor mName) {
     if (mName == null)
         return null;

     try {
         Object[] params = new Object[]{mName};
         String[] signature= new String[]{"com.ibm.websphere.pmi.stat.MBeanStatDescriptor"};
         MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke(perfOName, "listStatMembers",
   params, signature);
         if (msds == null)
             return null;
         for (int i=0; i<msds.length; i++) {
             MBeanStatDescriptor[] msds2 = listStatMembers(msds[i]);
             // you may recursively call listStatMembers until find the one you want
         }
         return msds;
```

```
        }
        catch(Exception e) {
            new AdminException(e).printStackTrace();
            System.out.println("listStatMembers: Exception Thrown");
            return null;
        }

    }

    /**
     * sample code to get PMI data from beanModule
     * @deprecated Use 6.0 APIs.
     */
    public void testEJB() {

        // This is the MBeanStatDescriptor for Enterprise EJB
        MBeanStatDescriptor beanMsd = getStatDescriptor(serverOName,
     PmiConstants.BEAN_MODULE);
        if (beanMsd == null)
            System.out.println("Error: cannot find beanModule");

        // get the Stats for module level only since recursive is false
        Stats stats = getStatsObject(beanMsd.getObjectName(), beanMsd.
getStatDescriptor(), false); // pass true if you wannt data from individual beans

        // find the avg method RT
        TimeStatisticImpl rt = (TimeStatisticImpl)stats.getStatistic
(EJBStatsImpl.METHOD_RT);
        System.out.println("rt is " + rt.getMean());

        try {
            java.lang.Thread.sleep(5000);
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        // get the  Stats again
        Stats stats2 = getStatsObject(beanMsd.getObjectName(), beanMsd.
getStatDescriptor(), false); // pass true if you wannt data from individual beans

        // find the avg method RT
        TimeStatisticImpl rt2 = (TimeStatisticImpl)stats2.getStatistic
(EJBStatsImpl.METHOD_RT);
        System.out.println("rt2 is " + rt2.getMean());

        // calculate the difference between this time and last time.
        TimeStatisticImpl deltaRt = (TimeStatisticImpl)rt2.delta(rt);
        System.out.println("deltaRt is " + rt.getMean());

    }

    /**
     * Sample code to show how to call getStats on StatisticProvider MBean
directly.
     * @deprecated Use 6.0 APIs.
     */
    public void testJSR77Stats() {
        // first, find the MBean ObjectName you are interested.
        // Refer method getObjectNames for sample code.

        // assume we want to call getStats on JVM MBean to get statistics
        try {

            com.ibm.websphere.management.statistics.JVMStats stats =
                    (com.ibm.websphere.management.statistics.JVMStats)ac.
    invoke(jvmOName, "getStats", null, null);
```

```
        System.out.println("\n get data from JVM MBean");

        if (stats == null) {
            System.out.println("WARNING: getStats on JVM MBean returns null");
        } else {

            // first, link with the static info if you care
            ((Stats)stats).setConfig(PmiClient.findConfig(configs, jvmOName));

            // print out all the data if you want
            //System.out.println(stats.toString());

            // navigate and get the data in the stats object
            processStats((Stats)stats);

            // call JSR77 methods on JVMStats to get the related data
            com.ibm.websphere.management.statistics.CountStatistic upTime =
  stats.getUpTime();
            com.ibm.websphere.management.statistics.BoundedRangeStatistic
  heapSize = stats.getHeapSize();

            if (upTime != null)
                System.out.println("\nJVM up time is " + upTime.getCount());
            if (heapSize != null)
                System.out.println("\nheapSize is " + heapSize.getCurrent());
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        new AdminException(ex).printStackTrace();
    }
}


/**
 Get PmiModuleConfig from server
 */
public PmiModuleConfig getStatsConfig (String statsType)
{
   try
   {
        return (PmiModuleConfig)ac.invoke(perfONname, "getConfig",
                                          new String[]{statsType},
                                          new String[]{"java.lang.String"});
   }
   catch(Exception e)
   {
        e.printStackTrace();
        return null;
   }
}

/**
 * Get PmiModuleConfig based on MBean ObjectName
  @deprecated Use com.ibm.websphere.pmi.client.PmiClient.findConfig()
 */
public PmiModuleConfig findConfig(ObjectName on) {
    if (on == null) return null;

    String type = on.getKeyProperty("type");
    System.out.println("findConfig: mbean type =" + type);

    for (int i=0; i<configs.length ; i++) {

        if (configs[i].getMbeanType().equals(type))
            return configs[i];
    }
    System.out.println("Error: cannot find the config");
```

```
        return null;

    }

    /**
     * Get PmiModuleConfig based on PMI module name
     @deprecated Use com.ibm.websphere.pmi.client.PmiClient.findConfig()
     */
    public PmiModuleConfig findConfig(String moduleName) {
        if (moduleName == null) return null;

        for (int i=0; i<configs.length ; i++) {

            if (configs[i].getShortName().equals(moduleName))
                return configs[i];
        }
        System.out.println("Error: cannot find the config");
        return null;

    }
}
```

Developing PMI interfaces (Version 4.0) (deprecated)

You can use the Performance Monitoring Infrastructure (PMI) interfaces to develop your own applications to collect and display performance information.

The Version 4.0 APIs are supported in this release, however, some data hierarchy changes have occurred in the PMI modules, including the enterprise bean and HTTP sessions modules. If you have an existing PmiClient application and you want to run it against Version 5.0, you might have to update the PerfDescriptor(s) based on the new PMI data hierarchy.

The getDataName and getDataId methods in PmiClient have also changed. They are now non-static methods in order to support multiple WebSphere Application Server versions. You might have to update your existing application which uses these two methods.

This section discusses the use of the Performance Monitoring Infrastructure (PMI) client interfaces in applications. The basic steps in the programming model follow:

1. Retrieve an initial collection or snapshot of performance data from the server. A client uses the CpdCollection interface to retrieve an initial collection or snapshot from the server. This snapshot, which is called Snapshot in this example, is provided in a hierarchical structure as described in data organization and hierarchy, and contains the current values of all performance data collected by the server. The snapshot maintains the same structure throughout the lifetime of the CpdCollection instance.
2. Process and display the data as specified. The client processes and displays the data as specified. Processing and display objects, for example, filters and GUIs, can register as CpdEvent listeners to data of interest. The listener works only within the same Java virtual machine (JVM). When the client receives updated data, all listeners are notified.
3. Display the new CpdCollection instance through the hierarchy. When the client receives new or changed data, the client can simply display the new CpdCollection instance through its hierarchy. When it is necessary to update the Snapshot collection, the client can use the update method to update Snapshot with the new data.

   ```
   Snapshot.update(S1);
   // ...later...
   Snapshot.update(S2);
   ```

Steps 2 and 3 are repeated through the lifetime of the client.

Compiling your monitoring applications

Use this page to find the JAR files required to compile your Performance Monitoring Infrastructure (PMI).

To compile your Performance Monitoring Infrastructure (PMI) code, you must have the following JAR file in your class path:

- %WAS_HOME%/plugins/com.ibm.ws.runtime_6.1.0.jar

The JAR files listed above are needed to compile the PmiJmxTest example application. If your monitoring applications use APIs in other packages, also include those packages on the class path. If any WebSphere Application Server class is not found with the above set of jars, then you can include all the WebSphere jars using:

```
javac -classpath %WAS_HOME%\plugins\com.ibm.ws.runtime_6.1.0.jar myclass.java
```

Running your new monitoring applications

Use this page to learn about the steps you must follow in order to run monitoring applications.

Follow these steps to run your monitoring applications.

1. You need a WebSphere Application Server installation or WebSphere Application Server J2EE client package to run a PMI application.
2. Use a PMI client API to write your own application.
3. Compile the newly-written PMI application and place it on the class path. (The jar files under %WAS_HOME%\lib and %WAS_HOME%\classes folder will be placed in the class path by the following script.)
4. To run a PMI application you need a WebSphere Application Server runtime environment (the application server installation or a J2EE client package). Using the following script to run the application:

   Note: The following is formatted for Windows based systems. You may need to adjust the script depending on your operating system.

   ```
   @echo off
   @setlocal

   call "%~dp0setupCmdLine.bat"

   "%JAVA_HOME%\bin\java" "%CLIENTSAS%" "%CLIENTSOAP%" -DwebsphereV5Statistics=false
   -Dwas.install.root="%WAS_HOME%" -Dws.ext.dirs="%WAS_EXT_DIRS%" -classpath "%WAS_CLASSPATH%"
   com.ibm.ws.bootstrap.WSLauncher com.ibm.websphere.pmi.PmiJmxTest %*
   ```

Performance Monitoring Infrastructure client package

Use this page to learn how to use the PmiClient application and JMX connector to communicate to the Perf MBean in an application server.

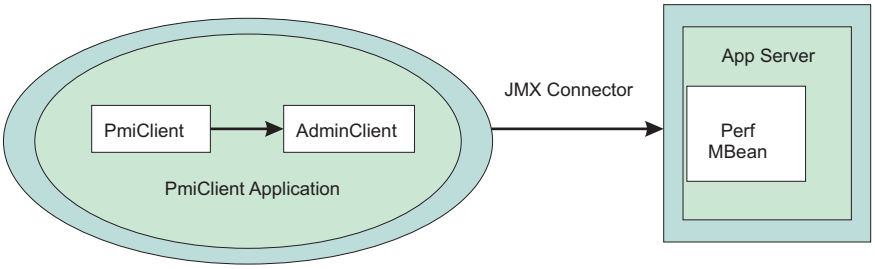
A Performance Monitoring Infrastructure (PMI) client package provides a PmiClient wrapper class to deliver PMI data to a client. As shown in the following figure, the PmiClient API uses the AdminClient API to communicate to the Perf MBean in an application server.

Performance Monitoring Infrastructure and Java Management Extensions

The PmiClient API does not work if the Java Management Extensions (JMX) infrastructure and Perf MBean are not running. If you prefer to use the AdminClient API directly to retrieve PMI data, you still have a dependency on the JMX infrastructure.

When using the PmiClient API, you have to pass the JMX connector protocol and port number to instantiate an object of the PmiClient. Once you get a PmiClient object, you can call its methods to list nodes, servers and MBeans, set the monitoring level, and retrieve PMI data.

The PmiClient API creates an instance of the AdminClient API and delegates your requests to the AdminClient API. The AdminClient API uses the JMX connector to communicate with the Perf MBean in the corresponding server and then returns the data to the PmiClient, which returns the data to the client.



Running your monitoring applications with security enabled

You can opt to run a Performance Monitoring Infrastructure client application with security enabled.

In order to run a Performance Monitoring Infrastructure (PMI) client application with security enabled, you must have %CLIENTSOAP% properties defined on your Java virtual machine command line for a SOAP connector, and %CLIENTSAS% properties defined for a RMI connector. The %CLIENTSOAP% and %CLIENTSAS% properties are defined in the setupCmdLine.bat or setupCmdline.sh files.

- If you are configuring security for a SOAP connector:
  1. Set com.ibm.SOAP.securityEnabled to **True** in the soap.client.props file for the SOAP connector. The soap.client.props property file is located in the *profile_root*/properties directory.
  2. Set com.ibm.SOAP.loginUserid and com.ibm.SOAP.loginPassword as the user ID and password for login.
- If you are configuring security for a RMI connector, set the sas.client.props file or type the user ID and password in the window, if you do not put them in the property file for Remote Method Invocation (RMI) connector. A common mistake is to leave extra spaces at the end of the lines in the property file. Do not leave extra spaces at the end of the lines, especially for the user ID and password lines.

Monitoring performance with Tivoli Performance Viewer (TPV)

Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console.

In Version 4.0, Tivoli Performance Viewer was named the Resource Analyzer. In Version 5.0, TPV is a standalone Java application. In Versions 6.0.x and later, TPV is embedded in the administrative console. From TPV, you can view current activity and summary reports, or log Performance Monitoring Infrastructure (PMI) performance data. TPV provides a simple viewer for the performance data collected by the Performance Monitoring Infrastructure.

By viewing TPV data, administrators can determine which part of the application and configuration settings to change in order to improve performance. For example, you can view the servlet summary reports, enterprise beans, and Enterprise JavaBeans (EJB) methods in order to determine what part of the application to focus on. Then, you can sort these tables to determine which of these resources has the highest response time. Focus on improving the configuration for those application resources taking the longest response time.

1. **Optional:** Adjust the Performance Monitoring Infrastructure (PMI) settings for the servers that you want to monitor. The PMI service is enabled by default with a basic set of counters enabled.
2. Monitor current server activity. You can view real-time data on the current performance activity of a server using TPV in the administrative console.
   - Use the performance advisors to examine various data while your application is running. The performance advisor in TPV provides advice to help tune systems for optimal performance by using collected PMI data.

- Configure user and logging settings for TPV. These settings may affect the performance of your application server.
- View summary reports on servlets, Enterprise JavaBeans (EJB) methods, connections pools and thread pools in WebSphere Application Server.
- View performance modules that provide graphs and of various performance data on system resources such as CPU utilization, on WebSphere pools and queues such as database connection pools, and on customer application data such as servlet response time. In addition to providing a viewer for performance data, TPV enables you to view data for other products or customer applications that have implemented custom PMI.

3. View server performance logs. You can record and view data that has been logged by TPV in the administrative console. Be sure to configure user and logging settings for TPV.
4. Log performance data. You can record real-time data for later retrieval and analysis.

Why use Tivoli Performance Viewer?

Administrators and programmers can monitor the overall health of WebSphere Application Server from within the administrative console by using Tivoli Performance Viewer (TPV).

From TPV, you can view current activity or log Performance Monitoring Infrastructure (PMI) performance data for the following:
- System resources such as CPU utilization
- WebSphere pools and queues such as a database connection pool
- Customer application data such as average servlet response time

You can also view data for other products or customer applications that implement custom PMI by using TPV. For more information on custom PMI, refer to “Enabling PMI data collection” on page 2212.

By viewing PMI data, administrators can determine which part of the application and configuration settings to alter in order to improve performance. For example, in order to determine what part of the application to focus on, you can view the servlet summary report, enterprise beans and Enterprise JavaBeans (EJB) methods, and determine which of these resources has the highest response time. You can then focus on improving the configuration for those application resources with the longest response times.

Tivoli Performance Viewer is used to help manage configuration settings by viewing the various graphs or using the Tivoli Performance Advisor. For example, by looking at the summary chart for thread pools, you can determine whether the thread pool size needs to be increased or decreased by monitoring the percent (%) usage. After configuration settings are changed based on the data provided, you can determine the effectiveness of the changes. To help with configuration settings, use the Tivoli Performance Advisor. The Advisor assesses various data while your application is running, and provides configuration setting advice to improve performance.

transition: In Version 4.0, Tivoli Performance Viewer was originally named the Resource Analyzer. In Version 5.0, TPV was a standalone Java application. In Versions 6.0.*x* and later, TPV is embedded in the administrative console.

TPV topologies and performance impacts

The Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console.

The following two topologies exist for the Tivoli Performance Viewer:
- Tivoli Performance Viewer running in a single server environment
- Tivoli Performance Viewer running in a Network Deployment environment

When the Tivoli Performance Viewer is running in a single server environment, the collection and viewing of the data occurs in the same Java virtual machine. Because the collection and viewing of data occurs in

the application server, performance may be affected depending on TPV settings. “Viewing Data with the Tivoli Performance Viewer” on page 2263 describes the various TPV settings and their effect on performance.

Related tasks

“Monitoring performance with Tivoli Performance Viewer (TPV)” on page 2258
Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console.

Viewing current performance activity

You can view the current performance activity of a server using the Tivoli Performance Viewer (TPV) in the administrative console.

Once monitoring is enabled, TPV monitors the performance activity of all servers on a node, which can include the following:

- Application servers
- Node agent for the node being monitored

TPV enables administrators and programmers to monitor the current health of WebSphere Application Server. Because the collection and viewing of data occurs in the application server, performance may be affected. To minimize performance impacts, monitor only those servers whose resources need to be optimized.

1. Click **Monitoring and Tuning > Performance Viewer > Current Activity** in the console navigation tree. The TPV current activity collection is displayed.
2. Start monitoring the current activity of a server in either of two ways:
   - Under **Server**, click the name of the server whose activity you want to monitor. Clicking on the name starts the monitoring for the server and displays the activity page for the server.
   - Select the check box for the server whose activity you want to monitor, and click **Start Monitoring**.

   A TPV console panel is displayed, providing a navigation tree on the left and a view of real-time data on the current performance activity of a server on the right.
3. From the navigation tree, select the server activity data that you want to view.

Option	Description
Advisor	Use the Performance Advisor to examine various data while your application is running. The Performance Advisor provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected PMI data.
Settings	Configure user and logging settings for TPV. These settings can affect the performance of your application server.
Summary Reports	View summary reports on servlets, enterprise beans (EJBs), EJB methods, connections pools and thread pools in WebSphere Application Server.
Performance Modules	View performance modules that provide graphics and charts of various performance data on system resources such as CPU utilization, on WebSphere pools and queues such as database connection pools, and on customer application data such as servlet response time. In addition to providing a viewer for performance data, TPV enables you to view data for other products or customer applications that have implemented custom PMI.

When you finish monitoring a server, select the server and click **Stop Monitoring**. TPV automatically stops monitoring a server if you the browser window is closed or if the user logs out..

Selecting a server and changing monitoring status

Use this page to start and stop monitoring for each server and to select a server for Tivoli Performance Viewer. You can also view the collection status for each server.

To view this administrative console page, click **Monitoring and Tuning** > **Performance Viewer** > **Current Activity**.

On this page, you can view the collection status of each server. When the collection status is **Monitored**, Tivoli Performance Monitor is enabled. If the collection status is **Available**, the server is ready to be enabled. If the collection status is **Unavailable, server stopped, PMI not enabled, or nodeagent stopped**, you must restart your server before you can start monitoring. Click on any server or node agent to view the current activity for that server. Lastly, **Performance Data Collection Not Supported** means that the server is not version 6.0.x or later.

Maximum rows: Specifies the maximum number of rows that displays when the collection is large. The rows that are not displayed appear on the next page.

Retain filter criteria: Specifies whether to use the same filter criteria entered in the show filter function to display this page the next time you visit it.

Start monitoring:

Select one or more servers from the list and press Start monitoring to start the Tivoli Performance Monitor for the selected servers.

Stop monitoring:

Select one or more servers from the list and press Stop monitoring to stop the Tivoli Performance Monitor for the selected servers.

Configuring TPV settings

You can configure user and logging settings of the Tivoli Performance Viewer (TPV). Configuring the TPV settings affects the performance of your application server.

TPV monitors the performance activity of all servers on a node.

The data can also be viewed from the deployment manager.

You can configure the activity monitoring of TPV on a per-user basis. Any changes made to TPV settings are only for the server being monitored and only affect the user viewing the data.

You can change the user and log TPV settings in the administrative console.

- Configure the TPV user settings.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Settings > User** in the console navigation tree. To see the **User** link on the Tivoli Performance Viewer page, expand the **Settings** node of the TPV navigation tree on the left side of the page. After clicking **User**, the TPV user settings are displayed on the right side of the page.
  2. Change the values as needed for the user settings. The settings are described briefly below and in more detail in the Tivoli Performance Viewer settings.

Refresh Rate	Specifies how frequently TPV collects performance data for a server from the Performance Monitoring Infrastructure (PMI) service provided by that server. The default is 30 seconds. To collect performance data for the server more frequently, set the refresh rate to a smaller number. To collect performance data less frequently, set the refresh rate to a larger number. The allowed range is 5 to 500 seconds.
Buffer Size	Specifies the number of entries to be stored for a server. Data displayed in TPV is stored in a short in-memory buffer. After the buffer is full, each time a new entry is retrieved the oldest entry is discarded. The default buffer size is 40 entries. Supported values are 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. The larger the buffer size, the more memory is consumed. Thus, specify a buffer size that allows you capture enough monitoring data for analysis without wasting memory storing unneeded data.
View Data As	Specifies how counter values are displayed. Viewing options include the following: **Raw Value** Displays the absolute value. If the counter represents load data, such as the average number of connections in a database pool, then TPV displays the current value followed by the average. For example, `18 (avg:5)`. **Change in Value** Displays the change in the current value from the previous value. **Rate of Change** Displays the ratio `change/(T1 - T2)`, where *change* is the change in the current value from the previous value, *T1* is the time when the current value was retrieved, and *T2* is the time when the previous value was retrieved.

The refresh rate and buffer size settings combine to control how much temporal history you have for the application server. The default values for **Refresh Rate** (30 seconds) and **Buffer Size** (40 entries) provide you with a 20-minute history of the application server's performance. Changing one of these parameters affects the length of the temporal history.

The values you set for **Refresh Rate** and **Buffer Size** depend on your use of TPV. To diagnose a known problem on a test machine, you might poll data more frequently while having a decreased buffer size. To monitor a production server, you might poll data less frequently and specify a buffer size depending on how much history you want. However, TPV is not intended to be a full-time monitoring solution.

3. Click **Apply**.

- Configure the TPV log settings. The log settings control what happens when **Start Logging** is clicked in, for example, a summary report on the performance of a servlet, enterprise bean (EJB), EJB method, connection pool or thread pool.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Settings > Log** in the console navigation tree. To see the **Log** link on the Tivoli Performance Viewer page, expand the **Settings** node of the TPV navigation tree on the left side of the page. After clicking **Log**, the TPV log settings are displayed on the right side of the page.
  2. Change the values as needed for the log settings. The settings are described below and in the Tivoli Performance Viewer settings.

Duration	Specifies the length of time, in minutes, that logging continues, unless **Stop Logging** is clicked first. TPV is not intended as a full-time logging solution.
Maximum File Size	Specifies the maximum size, in megabytes, of a single file. Note that TPV automatically zips log files to save space and this parameter controls the pre-zipped file size and not the post-zipped, which is smaller.
Maximum Number of Historical Files	Specifies the number of files TPV writes before stopping. If TPV reaches the maximum file size before the logging duration ends, it continues logging in another file, up to the maximum. If TPV reaches the maximum number of historical files before the logging duration ends, TPV deletes the oldest historical file and continues logging in a new file. The total amount of data that is stored is constrained by the **Maximum File Size** and **Maximum Number of Historical Files** parameters.
File Name	Specifies the name of the log file. The server name and the time at which the log is started is appended to the log name to help users identify a log file.
Log Output Format	Specifies whether TPV writes log files as XML or in a binary format. Binary format is recommended as it provides a smaller log file when uncompressed.

3. Click **Apply**.

Viewing Data with the Tivoli Performance Viewer

Use this page to view and refresh performance data for the selected server, change user and log settings, view summary reports, and information on specific performance modules.

To view this administrative console page, click **Monitoring and Tuning** > **Performance Viewer** > **Current Activity** > *server*.

Click the server name to view the current activity for that server. In this view, the Tivoli Performance Viewer (TPV) has two main parts, which include the navigation panel located beside the administrative console navigation tree, and the data viewing panel located to the right of the Tivoli Performance Viewer navigation panel.

Refresh:

Click **Refresh** to rebuild the navigation tree. Refreshing is helpful when the available Performance Monitoring (PMI) Infrastructure data has changed, and the tree does not reflect those changes.

View Module(s):

Click **View Module** after one or more performance modules are selected in the tree to display the information for these modules in the data viewing panel.

The Data Monitoring panel enables the selection of multiple counters and displays the resulting performance data for the associated resources. It consists of two panels:

- Viewing Counter panel
- Counter Selection panel. If necessary, you can change the scaling factors by editing the default values in the scale field.

Deselect all items:

Select **Deselect all items** to quickly deselect all modules that are selected in the navigation tree.

Navigation tree:

Use the navigation tree to view advisor output, configure TPV, and select PMI modules for viewing.

- Click the **Advisor** node to have the advisor display the data in the viewing panel.
- Expand the **Settings** node to select and configure either the **User** or **Log** settings.
- Expand the **Performance Modules** node to view one or more PMI modules.

Advisor:

Click **Advisor** to examine various data while your application is running. The Performance Advisor provides advice to help tune systems for optimal performance using the PMI data collected.

The first table represents the number of requests per second and the response time in milliseconds for the Web container.

The pie graph displays the CPU activity as a percentage of busy and idle.

The third table displays average thread activity for the different resources, for example, Default, Object Request Broker, and Web container. Activity is expressed as the number of threads or connections busy and idle.

To view detailed information about the advice, select the message you want to view. This view provides additional information about the advice message, severity, description, user action, and detail.

User settings:

Change the values as needed for the following user settings:

Refresh Rate	Specifies how frequently TPV collects performance data for a server from the Performance Monitoring Infrastructure (PMI) service provided by that server. The default is 30 seconds. To collect performance data for the server more frequently, set the refresh rate to a smaller number. To collect performance data less frequently, set the refresh rate to a larger number. The allowed range is 5 to 500 seconds.
Buffer Size	Specifies the amount of data to be stored for a server. Data displayed in TPV is stored in a short in-memory buffer. After the buffer is full, each time a new entry is retrieved the oldest entry is discarded. The default buffer size is 40. Allowed values are 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. The larger the buffer size, the more memory is consumed. Thus, specify a buffer size that allows you capture enough monitoring data for analysis without wasting memory storing unneeded data.

View Data As	Specifies how counter values are displayed. Viewing options include the following: **Raw Value** Displays the absolute value. If the counter represents load data, for example, the average number of connections in a database pool, then TPV displays the current value. **Change in Value** Displays the change in the current value from the previous value. **Rate of Change** Displays the ratio `change/(T1 - T2)`, where *change* is the change in the current value from the previous value, *T1* is the time when the current value was retrieved, and *T2* is the time when the previous value was retrieved.

Log settings:

The log settings control what happens when **Start Logging** is clicked in, for example, a summary report on the performance of a servlet, enterprise bean (EJB), EJB method, connection pool or thread pool.

Change the value as needed for the following log settings:

Duration	Specifies the length of time, in minutes, that logging continues, unless **Stop Logging** is clicked first. TPV is not intended as a full-time logging solution.
Maximum File Size	Specifies the maximum size, in megabytes, of a single file. Note that TPV automatically zips log files to save space and this parameter controls the pre-zipped file size and not the post-zipped, which is smaller.
Maximum Number of Historical Files	Specifies the number of files TPV writes before stopping. If TPV reaches the maximum file size before the logging duration ends, it continues logging in another file, up to the maximum. If TPV reaches the maximum number of historical files before the logging duration ends, TPV deletes the oldest historical file and continues logging in a new file. The total amount of data that is stored is constrained by the **Maximum File Size** and **Maximum Number of Historical Files** parameters.
File Name	Specifies the name of the log file. The server name and the time at which the log is started is appended to the log name to help users identify a log file.
Log Output Format	Specifies whether TPV writes log files as XML or in a binary format. Binary format is recommended as it provides a smaller log file when not compressed.

View summary reports:

Summary reports are available for each application server.

Before viewing reports, make sure data counters are enabled and monitoring levels are set properly.

The standard monitoring level enables all reports except the report on Enterprise JavaBeans (EJB) methods. To enable an EJB methods report, adjust the PMI level to include EJB method data.

Tivoli Performance Viewer provides the following summary reports for each application server:

Servlets

The servlet summary lists all servlets that are running in the current application server. Use the servlet summary view to quickly find the most time intensive servlets and the applications that use them, and to determine which servlets are invoked most often. You can sort the summary table by any of the columns.

Tips

- Sort by **Avg Response Time** to find the slowest servlet or JavaServer page (JSP).
- Sort by **Total Requests** to find the servlet or JSP used the most.
- Sort by **Total Time** to find the servlet or JSP with the highest response times.

Enterprise JavaBeans

The Enterprise JavaBeans (EJB) summary lists all enterprise beans running in the server, the amount of time spent in their methods, the number of EJB invocations, and the total time spent in each enterprise bean.

```
total_time = number_of_invocations * time_in_methods
```

Sort the various columns to find the most expensive enterprise bean. Also, if the PMI counters are enabled for individual EJB methods, select the check box next to the EJB name see statistics for each of the methods.

Tips

- Sort by **Avg Response Time** to find the slowest enterprise beam.
- Sort by **Method Calls** to find the enterprise bean used the most.
- Sort by **Total Time** to find the enterprise bean with the slowest response time.

EJB Methods

The EJB method summary shows statistics for each EJB method. Use the EJB method summary to find the most costly methods of your enterprise beans.

Tips

- Sort by **Avg Response Time** to find the slowest EJB method.
- Sort by **Method Calls** to find the EJB method used the most.
- Sort by **Total Time** to find the EJB method with the slowest response time.

Connection pools

The connection pool summary lists all data source connections that are defined in the application server and shows their usage over time.

Tip

- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

Thread pools

The thread pool summary shows the usage of all thread pools in the application server over time.

Tip

- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

Performance module:

View performance modules that provide graphics and charts of various performance data on system resources such as CPU utilization, on WebSphere Application Server pools and queues such as database connection pools, and on customer application data such as servlet response time. In addition to providing a viewer for performance data, TPV enables you to view data for other products or customer applications that have implemented custom PMI.

Each performance module has several counters associated with it. These counters are displayed in a table underneath the data chart or table. Selected counters are displayed in the chart or table. You can add or remove counters from the chart or table by selecting or deselecting the check box next to them. By default, the first three counters for each module are shown.

Tivoli Performance Viewer displays interactive graphics using the Scalable Vector Graphics (SVG) format or non-interactive graphics using the JPG format. The SVG format is recommended because it provides a better user experience and is more processor and memory efficient for the application server.

In the performance module, you view current activity. This is a real time operation where the state of various system resources and their usage is displayed. Unless logging is turned on, data generated in this scenario will not be saved and is unavailable for subsequent viewing and analysis. To monitor behavior and system resources, click **Start Logging**. The user can replay and analyze the file at a later time.

Start Logging/Stop Logging

Use this to start or stop logging performance data. Once you start monitoring for your server, you will be able to view real time operation in the TPV panels.

Reset to Zero

This sets a new baseline using the current counter readings at the instant the button is clicked. Future data points are plotted on the graph relative to their position at the time **Reset to Zero** is clicked. Data points gathered prior to the time **Reset to Zero** is clicked are not displayed, although they are still held in the TPV buffer. If **Undo Reset to Zero** is clicked again, TPV displays all data currently in the buffer from their original baseline, not from the **Reset to Zero** point.

View Table/View Graph

To view the data in a table, click **View Table** on the counter selection table. To toggle back to a chart, click **View Graph**.

Show Legend/Hide Legend

To view the legend for a chart, click **Show Legend**. To hide the legend, click **Hide Legend**.

Clear Buffer

To clear values from the table or chart, click **Clear Buffer** beneath the chart or table. This removes all PMI data.

Unable to view Scalable Vector Graphics on Internet Explorer: If you encounter problems while viewing SVG with your Internet Explorer browser, visit the Adobe SVG website at http://www.adobe.com/svg/main.html to test your Adobe SVG Viewer. You can also download the most recent version of the Adobe SVG Viewer, view the release notes, and report any bugs with the Adobe SVG Viewer from this Web site.

Viewing TPV summary reports

The Tivoli Performance Viewer (TPV) provides five different summary reports that make important data quickly and easily accessible. View summary reports to help you find performance bottlenecks in your applications and modules. The TPV summary reports are generated upon request and are not dynamically refreshed.

This article assumes that one or more applications or modules are deployed and running on one or more servers.

In order to prepare a specific summary report, you must enable the minimum level of PMI data collection, or utilize the **custom** level of monitoring and enable specific counters.

Use the administrative console to view TPV summary reports.

1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Summary Reports** in the console navigation tree.
2. Select the code artifact or pool for which you want a summary report. Expand the **Summary Reports** node of the TPV navigation tree on the left side of the Tivoli Performance Viewer page to see links for the types of summary reports. After clicking on a link for an artifact or pool, a list of artifacts or pools on the server is displayed on the right side of the page.
3. Select the artifact or pool for which you want to view a summary report.

Types of TPV Summary Reports:

Servlets

The servlet summary lists all servlets that are running in the current application server. Use the servlet summary view to quickly find the most time intensive servlets and the applications that use them, and to determine which servlets are invoked most often. You can sort the summary table by any of the columns.

Tips

- Sort by **Avg Response Time** to find the slowest servlet or JavaServer page (JSP).
- Sort by **Total Requests** to find the servlet or JSP used the most.
- Sort by **Total Time** to find the most costly servlet or JSP.

Enterprise beans

The Enterprise JavaBeans (EJB) summary lists all enterprise beans running in the server, the amount of time spent in their methods, the number of EJB invocations, and the total time spent in each enterprise bean.

```
total_time = number_of_invocations * time_in_methods
```

Sort the various columns to find the most expensive enterprise bean. Also, if the PMI counters are enabled for individual EJB methods, there is a check box next to the EJB name that you can select to see statistics for each of the methods.

Tips

- Sort by **Avg Response Time** to find the slowest enterprise beam.
- Sort by **Method Calls** to find the enterprise bean used the most.
- Sort by **Total Time** to find the most costly enterprise bean.

EJB methods

The EJB method summary shows statistics for each EJB method. Use the EJB method summary to find the most costly methods of your enterprise beans.

Tips

- Sort by **Avg Response Time** to find the slowest EJB method.
- Sort by **Method Calls** to find the EJB method used the most.
- Sort by **Total Time** to find the most costly EJB method.

Connection pools

The connection pool summary lists all data source connections that are defined in the application server and shows their usage over time.

Tip
- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

Thread Pools

The thread pool summary shows the usage of all thread pools in the application server over time.

Tip
- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

PMI levels and counters required

In order to view Tivoli Performance Viewer (TPV) summary reports, the minimum PMI level must be enabled. Otherwise, you must use the **custom** monitoring level, and enable the PMI level counters required for the specific report you want to view.

Table 63. Required properties for TPV summary reports

Summary Report	PMI level required	Custom PMI counters required
Servlets	Basic	JDBC Connection Pools.PoolSize JDBC Connection Pools.AllocateCount JDBC Connection Pools.ReturnCount
Enterprise beans	Basic	Thread Pools.PoolSize Thread Pools.ActiveCount
EJB methods	All	Enterprise Beans.MethodCallCount Enterprise Beans.MethodResponseTime
Connection pools	Extended	WSEJBStats.MethodStats.MethodLevelCallCount WSEJBStats.MethodStats.MethodLevelResponseTime
Thread pools	Extended	Web Applications.RequestCount Web Applications.ServiceTime

Related tasks

"Enabling PMI data collection" on page 2212
Enable PMI data collection to diagnose problems and tune application performance.

"Viewing TPV summary reports" on page 2268
The Tivoli Performance Viewer (TPV) provides five different summary reports that make important data quickly and easily accessible. View summary reports to help you find performance bottlenecks in your applications and modules. The TPV summary reports are generated upon request and are not dynamically refreshed.

Viewing PMI data with TPV

You can use the Tivoli Performance Viewer (TPV) to view Performance Monitoring Infrastructure (PMI) data in chart or table form.

TPV monitors the performance activity of all servers on a node. This article assumes that one or more servers have been created and are running on the node, and that PMI is enabled.

TPV displays graphics in either the Scalable Vector Graphics (SVG) format or as a static image in the JPG format. If you do not have the Adobe SVG browser plug-in installed, you are prompted to download and install it. If you select not to install the plug-in (by selecting **Cancel**), TPV displays the static image. Installing the Adobe SVG plug-in is advantageous for several reasons. First, the SVG format provides interactive graphics that provide additional information when you hover your mouse over a point, line, or legend item. The SVG format also enables you to click a point and see details for it. Secondly, using the SVG format provides a performance benefit because the work to display the SVG image is done on the client side. When viewing a static image, the application server must convert the SVG image into a static image, which is a processor and memory intensive operation.

If you experience problems with SVG, see "Problems with Scalable Vector Graphics" on page 2271 for more information.

You view performance modules when your server is experiencing performance problems. For example, a common performance problem occurs when individual sessions are too large. To help view data on a session, you can view the Servlet Session Manager PMI module and monitor the **SessionObjectSize** counter to make sure that **Session Object Size** is not too large.

Performance modules are shown in the TPV current activity settings in the administrative console.

- Select PMI data to view.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Performance Modules** in the console navigation tree.
  2. Place a check mark in the check box beside the name of each performance module that you want to view. Expand the tree by clicking **+** next to a node and shrink it by clicking **–** next to a node.

     If you do not see all the PMI counters you expect, or a PM you are interested in is visible but cannot be selected, PMI is not enabled for that particular counter or for any counter in that PM. Go to the PMI control area of the administrative console and enable PMI for any counters you wish to view, then return to TPV and select those PMs. To view the PMI page, click on **Monitoring and Tuning > Performance Monitoring Infrastructure >** *server_name*.
  3. Click on **View Modules**. A chart or table providing the requested data is displayed on the right side of the page. Charts are displayed by default.

     Each module has several counters associated with it. These counters are displayed in a table underneath the data chart or table. Selected counters are displayed in the chart or table. You can add or remove counters from the chart or table by selecting or deselecting individual counters. By default, the first three counters for each module are shown.

     You can select up to 20 counters and display them in the TPV in the **Current Activity** mode. The data from all the PMI counters that are currently enabled in each PM for all active instances of the PM can be recorded and captured in a TPV log file. Refer to "Logging performance data with TPV" on page 2272 for more information on the TPV log file. You can view the TPV log file for a particular

time period multiple times, selecting different combinations of up to 20 counters each time. You have the flexibility to observe the relationships among different performance data in a server during a particular period of time.

4. **Optional:** To remove a module from a chart or table, deselect the check box next to the module and click **View Modules** again.
5. **Optional:** To view the data in a table, click **View Table** on the counter selection table. To toggle back to a chart, click **View Graph**.
6. **Optional:** To view the legend for a chart, click **Show Legend**. To hide the legend, click **Hide Legend**.

- Scale the PMI data. You can manually adjust the scale for each counter so that the graph displays meaningful comparisons of different counters.
  1. Find the counter whose scale you want to modify in the table beneath the chart.
  2. Change the value for **Scale** as needed. **Tips:**
     – When the scale is set to 1 the true value of the counter is displayed in the graph.
     – A value greater than 1 indicates that the value is amplified by the factor shown.
     – A value less than 1 indicates that the variable is decreased by the factor shown.

     For example, a scale setting of .5 means that the counter is graphed as one-half its actual value. A scale setting of 2 means that the counter is graphed as twice its actual value. Scaling only applies to the graphed values and has no effect on the data displayed when viewing it in table form.
  3. Click **Update**.
- Clear values from tables and charts.
  1. Ensure that one or more modules are selected under **Performance Modules** in the TPV navigation tree
  2. Click **Clear Buffer** beneath the chart or table. The PMI data is removed from a table or chart.
- Reset counters to zero (0).
  1. Ensure that one or more modules are selected under **Performance Modules** in the TPV navigation tree
  2. Click **Reset to Zero** beneath the chart or table. **Reset to Zero** sets a new baseline using the current counter readings at the instant the button is clicked. Future datapoints are plotted on the graph relative to their position at the time **Reset to Zero** is clicked. Datapoints gathered prior to the time **Reset to Zero** is clicked are not displayed, although they are still held in the TPV buffer. If **Undo Reset to Zero** is clicked again, TPV displays all data recorded from the original baseline, not from the **Reset to Zero** point.

     For information on how TPV displays the data, see “Configuring TPV settings” on page 2261.

Problems with Scalable Vector Graphics:

Consult this topic for problems and solutions for Scalable Vector Graphics.

Using TPV without Scalable Vector Graphics (SVG)

When viewing performance data in Tivoli Performance Viewer (TPV) on a browser that does not support Scalable Vector Graphics (SVG) graphics, either using the Adobe SVG plug-in or through native support, complex graphs might take a long time to refresh. Eventually an out of memory error may occur.

When TPV detects a browser that does not support SVG, it uses Apache Batik to transcode the SVG into a binary image format. The transcoding process is highly memory intensive. The complexity of the graph is affected by two things: the TPV buffer size, which dictates how many points are in each line that is drawn, and the number of selected data points, which dictates how many lines are drawn.

To avoid long refresh times or the out of memory errors, use a browser that supports SVG, either with a plug-in, like the Adobe SVG plug-in, or natively, which is included in some Mozilla builds. If this is not

possible, keep the graph simple by limiting the size of the buffer and the number of selected data points.

Unable to view Scalable Vector Graphics on Internet Explorer

If you encounter problems while viewing SVG with your Internet Explorer browser, visit the Adobe SVG Zone Web site at http://www.adobe.com/svg/main.html to test your Adobe SVG Viewer. You can also download the most recent version of the Adobe SVG Viewer, view the release notes, and report any bugs with the Adobe SVG Viewer from this Web site.

Logging performance data with TPV

The Tivoli Performance Viewer (TPV) provides an easy way to store real-time data for system resources, WebSphere Application Server pools and queues, and applications in log files for later retrieval. You can start and stop logging while viewing current activity for a server, and later replay this data. Logging of performance data captures performance data in windows of time so you can later analyze the data.

This article assumes that one or more servers have been created and are running on the node, and that you have configured the TPV log settings. The log settings may affect performance and are described in detail in “Viewing Data with the Tivoli Performance Viewer” on page 2263. The TPV logging feature is not intended to be a full-time monitoring solution, but instead for selective data recording for subsequent replay and analysis.

You can study the sequence of events that led to a peculiar condition in the application server or node agent. First, enable TPV logging so performance data generated in the application server persists in a log file stored at a specific location. Later, using the replay feature in TPV, view the performance data that was generated in exactly the same chronological order as it was generated in real time, enabling you to analyze a prior sequence of events.

You do not need to know the syntax and format in which log files are generated and stored. Do not edit log files generated by TPV; doing so will irrecoverably corrupt or destroy the performance data stored in the log files.

You can create and view logs in the administrative console.

- Create logs.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Settings > Log** in the console navigation tree. To see the **Log** link on the Tivoli Performance Viewer page, expand the **Settings** node of the TPV navigation tree on the left side of the page. After clicking **Log**, the TPV log settings are displayed on the right side of the page.
  2. Click on **Start Logging** when viewing summary reports or performance modules.
  3. When finished, click **Stop Logging** . Once started, logging stops when the logging duration expires, **Stop Logging** is clicked, or the file size and number limits are reached. To adjust the settings, see step 1.

  By default, the log files are stored in the *profile_root*/`logs/tpv` directory on the node on which the server is running. TPV automatically compresses the log file when it finishes writing to it to conserve space. At this point, there must only be a single log file in each `.zip` file and it must have the same name as the `.zip` file.
- View logs.
  1. Click **Monitoring and Tuning > Performance Viewer > View Logs** in the console navigation tree.
  2. Select a log file to view using either of the following options:
     Explicit Path to Log File
     Choose a log file from the machine on which the browser is currently running. Use this option if you have created a log file and transferred it to your system. Click **Browse** to open a file browser on the local machine and select the log file to upload.

Server File

Specify the path of a log file on the server.

In a stand-alone application server environment, type in the path to the log file. The *profile_root*\logs\tpv directory is the default on a Windows system.

3. Click **View Log**. The log is displayed with log control buttons at the top of the view.
4. Adjust the log view as needed. Buttons available for log view adjustment are described below. By default, the data replays at the **Refresh Rate** specified in the user settings. You can choose one of the **Fast Forward** modes to play data at rate faster than the refresh rate.

Rewind	Returns to the beginning of the log file.
Stop	Stops the log at its current location.
Play	Begins playing the log from its current location.
Fast Forward	Loads the next data point every three (3) seconds.
Fast Forward 2	Loads ten data points every three (3) seconds.

You can view multiple logs at a time. After a log has been loaded, return to the View Logs panel to see a list of available logs. At this point, you can load another log.

TPV automatically compresses the log file when finishes writing it. The log does not need to be decompressed before viewing it, though TPV can view logs that have been decompressed.

Viewing Data Recorded by the TPV

Use this page to view logged data from Tivoli Performance Viewer.

To view this administrative console page, click **Monitoring and Tuning** > **Performance Viewer** > **View logs**.

Explicit path to log file:

Select explicit path to a log file, specify the path name, and click **View log** to display the stored data.

Server file:

Select server file, specify the path name, and click **View log** to display the stored data.

Third-party performance monitoring and management solutions

Several performance monitoring, problem determination, and management solutions are available that can be used with WebSphere Application Server.

These products use WebSphere Application Server interfaces, including:

- Performance Monitoring Infrastructure (PMI)
- Java Management Extensions (JMX)
- Request metrics Application Response Measurement (ARM)

See the topic “Performance: Resources for learning” on page 2097 for a link to IBM business partners providing monitoring solutions for WebSphere Application Server.

Chapter 25. Monitoring application flow

Monitoring, optimizing, and troubleshooting WebSphere Application Server performance can be a challenge. This article gives you a basic strategy for monitoring with an understanding of the application view.

This information includes understanding the application flow that satisfies the end user request. This perspective provides the views of specific servlets that access specific session beans, entity container-managed persistence beans, and a specific database. This perspective is important for the in-depth internal understanding of who is using specific resources. Typically at this stage, you deploy some type of trace through the application, or thread analysis under load condition techniques to isolate areas of the application and particular interactions with the back-end systems that are especially slow under load. In this case, WebSphere Application Server provides request metrics to help trace each individual transaction as it flows through the application server, recording the response time at different stages of the transaction flow (for example, request metrics records the response times for the Web server, the Web container, the Enterprise JavaBeans container, and the back-end database). In addition, several IBM development and monitoring tools that are based on the request metrics technology (for example, Tivoli Monitoring for Transaction Performance) are available to help view the transaction flow.

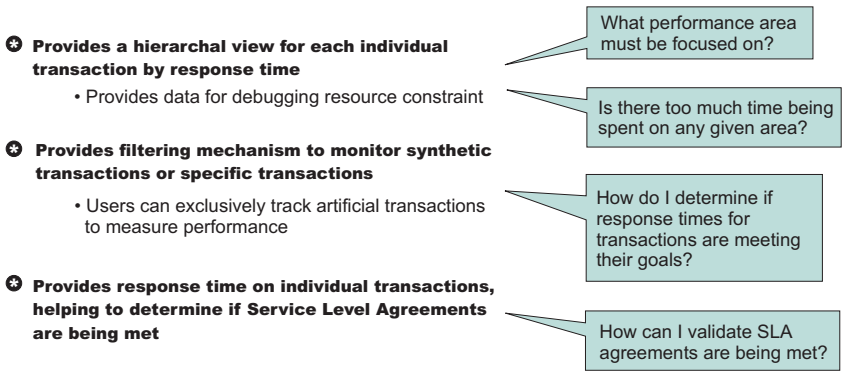
Why use request metrics?

Request metrics is a tool that enables you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components.

The information that is tracked by request metrics might either be saved to log files for later retrieval and analysis, be sent to Application Response Measurement (ARM) agents, or both.

As a transaction flows through the system, request metrics includes additional information so that the log records from each component can be correlated, building up a complete picture of that transaction. The result looks similar to the following example:

```
HTTP request/trade/scenario ------------------------------> 172 ms
     Servlet/trade/scenario  ------------------------------> 130 ms
         EJB TradeEJB.getAccountData ---------------------->  38 ms
              JDBC select ------------------------------->   7 ms
```



This transaction flow with associated response times can help you target performance problem areas and debug resource constraint problems. For example, the flow can help determine if a transaction spends most of its time in the Web server plug-in, the Web container, the Enterprise JavaBeans (EJB) container or the back end database. The response time that is collected for each level includes the time spent at that level and the time spent in the lower levels. For example, the response time for the servlet, which is 130

milliseconds, also includes 38 milliseconds from the enterprise beans and Java Database Connectivity. Therefore, 92 ms can be attributed to the servlet process.

Request metrics tracks the response time for a particular transaction. Because request metrics tracks individual transactions, using it imposes some performance implications on the system. However, this function can be mitigated by the use of the request filtering capabilities.

For example, tools can inject synthetic transactions. Request metrics can then track the response time within the WebSphere Application Server environment for those transactions. A synthetic transaction is one that is injected into the system by administrators to take a proactive approach to testing the performance of the system. This information helps administrators tune the performance of the Web site and take corrective actions. Therefore, the information that is provided by request metrics might be used as an alert mechanism to detect when the performance of a particular request type goes beyond acceptable thresholds. The filtering mechanism within request metrics might be used to focus on the specific synthetic transactions and can help optimize performance in this scenario.

When you have the isolated problem areas, use request metrics filtering mechanism to focus specifically on those areas. For example, when you have an isolated problem in a particular servlet or EJB method, use the uniform resource identifier (URI) algorithms or EJB filter to enable the instrumentation only for the servlet or EJB method. This filtering mechanism supports a more focused performance analysis.

Five types of filters are supported:
- Source IP filter
- URI filter
- EJB method name filter
- JMS parameters filter
- Web services parameters filter

When filtering is enabled, only requests that match the filter generate request metrics data, create log records, call the ARM interfaces, or all. You can inject the work into a running system (specifically to generate trace information) to evaluate the performance of specific types of requests in the context of a normal load, ignoring requests from other sources that might be hitting the system.

Note: Filters are only applicable where the request first enters WebSphere Application Server.
- Learn more about request metrics by reviewing detailed explanations in this section.
- Configure and enable request metrics.
- Retrieve performance data and monitor application flow.
- Extend monitoring to track applications that might require additional instrumentation points.

Example: Using request metrics

Use this page to learn how to use request metrics by viewing an example.

In this example, the HitCount servlet and the Increment enterprise bean are deployed on two different application server processes. As shown in the following diagram, the Web container tier and Enterprise JavaBeans (EJB) container tiers are running in two different application servers. To set up such a configuration, install WebSphere Application Server Network Deployment.

Assume that the Web server and the Web container tier both run on machine 192.168.0.1, and the Enterprise JavaBeans (EJB) container tier runs on a second machine 192.168.0.2. The client requests might be sent from a different machine; 192.168.0.3, for example, or other machines.

To illustrate the use of source IP filtering, one source IP filter (192.168.0.3) is defined and enabled. You can trace requests that originate from machine 192.168.0.3 through http://192.168.0.1/hitcount?selection=EJB&lookup=GBL&trans=CMT. However, requests that originate from any other machines are not traced because the source IP address is not in the filter list.

By only creating a source IP filter, any requests from that source IP address are effectively traced. This tool is effective for locating performance problems with systems under load. If the normal load originates from other IP addresses, then its requests are not traced. By using the defined source IP address to generate requests, you can see performance bottlenecks at the various hops by comparing the trace records of the loaded system to trace records from a non-loaded run. This ability helps focus tuning efforts to the correct node and process within a complex deployment environment.

Make sure that request metrics is enabled using the administrative console. Also, make sure that the trace level is set to at least hops (writing request traces at process boundaries). Using the configuration previously listed, send a request http://192.168.0.1/hitcount?selection=EJB&lookup=GBL&trans=CMT through the HitCount servlet from machine 192.168.0.3.

In this example, at least three trace records are generated:

- A trace record for the Web server plug-in is displayed in the plug-in log file (default location is (*plugins_root*/logs/*web_server_name*/http_plugin.log) on machine 192.168.0.1.
- A trace record for the servlet displays in the application server log file (default location is *profile_root*/logs/appserver/SystemOut.log) on machine 192.168.0.1.
- A trace record for the increment bean method invocation displays in the application server log file (default location is *profile_root*/logs/appserver/SystemOut.log) on machine 192.168.0.2.

The two trace records that are displayed on machine 192.168.0.1 are similar to the following example:

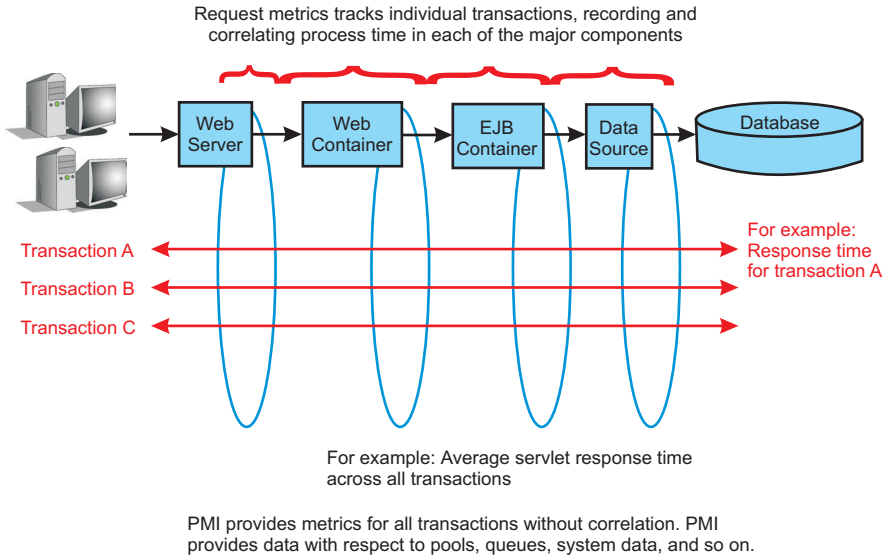
```
PLUGIN:
parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=0
- current:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=1
type=HTTP detail=/hitcount elapsed=90 bytesIn=0 bytesOut=2252

Application server (web container tier)
PMRM0003I: parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=0
- current:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
type=URI detail=/hitcount elapsed=60
```

The trace record that is displayed on machine 192.168.0.2 is similar to the following example:

```
PMRM0003I:
parent:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
-
current:ver=1,ip=192.168.0.2,time=1016556190505,pid=9321,reqid=40,event=1
type=EJB
detail=com.ibm.defaultapplication.Increment.increment elapsed=40
```

Data you can collect with request metrics

Typically, different components of the enterprise application might be hosted across several nodes in a distributed system. For example, the servlets might be hosted on one node, while the enterprise beans on which these servlets depend might be hosted on an entirely different node. When a request comes to a process, the process might send the request to one or more downstream processes, as shown in the following figure:



Trace records might be generated for each process with associated elapsed times for that process. These trace records might be correlated together to build a complete picture of the request flow through the distributed system, similar to the diagram in “Why use request metrics?” on page 2275.

You can view the process response time that is monitored by request metrics through the Application Response Measurement (ARM) interface and system log files. When a request is sent to the application server, request metrics captures response times for the initiating request and any related downstream invocations. Request metrics are instrumented in the following components as the request, for example, transaction, travels through the Web server, Proxy Server and the application server:

- the Web server plug-in which is only available when using the Web server port.
- the Proxy Server, instrumented as servlet and Web services requests.
- the Web container, including servlet and servlet filters.
- the Enterprise JavaBeans (EJB) container.
- Java DataBase Connectivity (JDBC) calls.
- J2EE Connector Architecture (JCA).
- Web services, both on the server and the client side.
- the Java Message Service (JMS) engine.
- Service Integration Bus (SIB).
- Portlet container, including the portlet requests.
- Asynchronous Beans.

Select which components that you want to instrument. For example, if you want instrumentation data only for the Web container and the JMS API, select this data in the administrative console and the detailed instrumentation data is generated only for the components that you select. The edge transactions are traced for the other components that are not specified for instrumentation.

When filtering is enabled, only requests that match the filter generate request metrics data, create log records, or call the ARM interfaces. You can add work into a running system specifically to generate trace information to evaluate the performance of specific types of requests in the context of normal load, ignoring requests from other sources that might affect the system. If the request matches any filter with a trace level greater than None, trace records are generated for that request.

Getting performance data from request metrics

This topic describes how to enable request metrics.

Request metrics is a tool that enables you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components.

You can enable request metrics from the following locations:

- The administrative console. To enable request metrics in the administrative console, refer to the instructions under the Steps for this task section that follows.
- Command line. Java Management Extensions (JMX) interfaces are exposed for enabling request metrics through external tools. For more details on the exposed interfaces, refer to the request metrics API documentation.

To enable request metrics in the administrative console:

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request metrics** in the console navigation tree.
3. Verify that **Prepare servers for request metrics collection** check box is selected. If **Prepare servers for request metrics collection** check box is not selected, you will have to select it, save the change, and restart the server.
4. Select All, None, or Custom under **Components to be instrumented** to specify the request metrics components. If you select **Custom**, be sure to select the components you would like to enable.
5. Specify the components that are instrumented by request metrics.
6. Specify how much data to collect.
7. Enable and disable logging.
8. Enable Application Response Measurement (ARM) Agent.
9. Specify which ARM type to use.
10. Specify the name of the ARM transaction factory implementation class.
11. Isolate performance for specific types of requests.
    a. Add and remove request metrics filters.
12. Click **Apply** or **OK**.
13. Click **Save**.

The request metrics is enabled.

To ensure that the Web server plug-in recognizes the changes you made for the request metrics configuration, follow the steps in “Regenerating the Web server plug-in configuration file” on page 2292, if logging time spent in the Web server.

Request metrics

Use this page to enable request metrics, select the components that are instrumented by request metrics, set trace levels, enable standard logs, enable Application Response Measurement (ARM), specify the type of ARM agent, and specify the ARM transaction factory implementation class name.

To view this administrative console page, click **Monitoring and Tuning** > **Request metrics**.

Prepare Servers for request metrics collection

Turns on the request metrics feature.

When unchecked, the request metrics function is disabled. To enable request metrics, check the box, save the change, and restart the server. When it is checked, the server is ready to enable request metrics and

you will not need to restart the server. Depending on what option is selected under **Components to be instrumented**, request metrics instrumentation will be enabled/disabled in various components.

Note: This selection process differs from the **Enable Request Metrics** check box in WebSphere Application Server Version 6.0x. You now have the option of selecting All, None, or Custom. If you select **Custom**, you must specify which components you would like to enable.

Components to be instrumented

Selects the components that are instrumented by request metrics.

When **None** is selected, no request metrics instrumentation will be enabled. When **All** is selected, request metrics instrumentation will be enabled in all the components listed under **Custom**. When **Custom** is selected, request metrics instrumentation will be enabled in the selected components.

Note: An edge transaction, which is defined as the first transaction that enters the application server without a parent correlator, will always be instrumented even if the corresponding component is disabled for instrumentation.

Trace level

Specifies how much trace data to accumulate for a given transaction. Note that **Trace level** and **Components to be instrumented** work together to control whether or not a request will be instrumented.

Including one of the following values:

None No instrumentation.

Hops Generates instrumentation information on process boundaries only (for example, at the entry and exit points for the Web container).

Generates instrumentation information on process boundaries only (for example, a servlet request coming from a browser or a Web server and a JDBC request going to a database).

Performance_debug

Generates one additional level of instrumentation data, whereas debug generates detailed instrumentation data.

Generates the data at Hops level and the first level of the intra-process servlet and Enterprise JavaBeans (EJB) call (for example, when an inbound servlet forwards to a servlet and an inbound EJB calls another EJB). Other intra-process calls like naming and service integration bus (SIB) are not enabled at this level.

Debug

Provides detailed instrumentation data, including response times for all intra-process servlet and Enterprise JavaBeans (EJB) calls.

Provides detailed instrumentation data, including response times for all intra-process calls.

Note: Requests to servlet filters will only be instrumented at this level.

Standard logs

Enables the request metrics logging feature.

Select this check box to trigger the generation of request metrics logs in the `SystemOut.log` file.

Note: Since enabling the request metrics logging feature will increase processor usage, it is recommended using this feature together with filters so that only selected requests are instrumented.

Application Response Measurement (ARM) agent

Enables request metrics to call an underlying Application Response Measurement (ARM) agent.

Before enabling ARM, you need to install an ARM agent and configure it to the appropriate classpath and path, following the instructions of the ARM provider.

Specify ARM agent
Specifies the type of ARM agent that you want to use.

The ARM 4.0 agent and Tivoli ARM 2.0 agent are supported.

ARM transaction factory implementation class name
Specifies the ARM transaction factory implementation class name in the package that is supplied by your provider. This field is required when ARM 4.0 agent is selected, but it is not required when Tivoli ARM agent is selected.

In this field, type the name of the ARM transaction factory implementation class that is present in the ARM library. Be sure to follow the instructions of the ARM provider and understand the name of the ARM transaction factory class for the installed ARM agent.

Filters
When filtering is enabled, only requests matching the specified filter will generate request metrics data. Filters exist for source IP address, URI name, EJB method name, JMS parameters, and the WebServices parameters.

Application Response Measurement
Request metrics information might be either saved to the log file for later retrieval and analysis, be sent to Application Response Measurement (ARM) agents, or both. Request metrics provides response time for each of the major WebSphere Application Server components through ARM APIs.

ARM is an Open Group standard. Request metrics helps you to plug in an ARM agent to collect response time measurements.

WebSphere Application Server does not ship an ARM agent. However, it supports the use of agents adhering to ARM 4.0 and ARM 2.0 standards.

You can choose your own ARM implementation providers to obtain the ARM implementation libraries. Follow the instruction from the ARM provider, and ensure that the ARM API Java archive (JAR) files found in the ARM provider are on the class path so that the WebSphere Application Server can load the needed classes. In the case of Tivoli Monitoring Transaction Performance, V5.3, copy the `armjni.jar` and `core_util.jar` files from the Tivoli Monitoring Transaction Performance *<tmtp_install_root>*`/lib` installation root directory to the *app_server_root*`/lib` directory, which is the WebSphere Application Server installation root directory. If the underlying ARM implementation is ARM 4.0, you need to specify the ARM transaction factory class name. Otherwise, this specification is not required.

See the article “Performance: Resources for learning” on page 2097 for more information about the ARM specifications.

ARM application properties and transaction context data
Request metrics provides build-in instrumentation to monitor transaction flows. The data that is collected by request metrics can be sent to a supported Application Response Measurement (ARM) agent.

ARM application properties

The following tables show the ARM application properties when request metrics initializes an ARM 4.0 agent.

Identity property names	Value

Cell Name	The cell name that the server belongs to.
Version	The version of WebSphere Application Server.

Application Properties	**Value**
Registered application name	WebSphere:<server_type> The <server_type> can be APPLICATION_SERVER, ONDEMAND_ROUTER, or PROXY_SERVER.
Application group	<server_name>
Application instance	<node_name>.<server_name>

The following code sample displays how WebSphere Application Server creates an ARM application.

```
String serverType; // the server type like APPLICATION_SERVER
String version; // WebSphere version
String sCellName; // the cell name of dMgr
String sAppInstance. // <short_node_name>.<short_server_name>
String sServerInstance; // short server name
String sWasName = "WebSphere:" + serverType;
String[] IDNAMES = new String[]{"Cell Name", "Version"};
ArmIdentityProperties appIdentity = txFactory.newArmIdentityProperties(IDNAMES, new String[]{sCellName,
version}, null);
ArmApplicationDefinition appDef = txFactory.newArmApplicationDefinition( sWasName, appIdentity, null );
ArmApplication app = txFactory.newArmApplication(appDef, sServerInstance, sAppInstance, null );
```

ARM transaction types

You can use the following code sample to create an instance of ARM transaction.

```
String[] contextNames; // the names for the context data like Port, QueryString, URI, EJBName
String tranIdentityName; // the transaction types shown in the following table like URI, EJB.
String appDef; // defined and created in above code snippet under ARM application properties
String app; // defined and created in above code snippet under ARM application properties
ArmIdentityPropertiesTransaction props = armFactory.newArmIdentityPropertiesTransaction( null, null,
contextNames, null );
ArmTransactionDefinition atd = armFactory.newArmTransactionDefinition(appDef, identityName, props, (ArmID)null);
ArmTransaction at = txFactory.newArmTransaction(app, atd );
```

The sections below show all ARM transaction types and their corresponding context names. Some types of transactions are not instrumented unless the trace level is set to DEBUG. In addition, not every transaction is available in all types of servers. All transaction types and context names are case sensitive.

• “Portlet” on page 2287

Uniform resource identifier (URI)

This transaction type is the uniform resource identifier (URI) for servlet and JavaServer Page (JSP) requests. All of the following transaction types and context names are available in all types of servers.

Transaction type: URI	
Context name	**Description**
Port	The TCP/IP port where the request arrived, specified as a string representation of the decimal value. **Example:** 9080
QueryString	The portion of the dynamic URI that contains the search parameters of the request in the query segment of the URI string, excluding the question mark (?) character. **Example:** selection=EJB&lookup=GBL&trans=CMT
URI	The incoming request URI of the servlet and JSP file used. **Example:** /hitcount

Enterprise Java Bean (EJB)

This EJB transaction type and context names are only available in application servers.

Transaction Type: EJB	
Context name	**Description**
EJBName	The fully qualified Enterprise Java™ Bean (EJB) class name, followed by method name with a period (.) to connect them. For example, com.mypackge.MyEJBClass.mymethod. **Example:** com.ibm.defaultapplication.IncrementBean.create
ApplicationName	The Java™ 2 Platform, Enterprise Edition (J2EE) application name that contains the enterprise bean. **Example:** DefaultApplication
ModuleName	The J2EE module name that contains the enterprise bean. **Example:** Increment.jar

Servlet filter

The servlet filter transaction type is only available at the DEBUG trace level. For other levels, servlet filters are not instrumented. This transaction type and context names are only available in application servers.

Transaction Type: Servlet filter	
Context name	**Description**

Transaction Type: Servlet filter	
FilterName	The servlet filter name for the servlet. **Example:** ALoginFilter

Java Database Connectivity (JDBC)

The JDBC transaction type is only available at the DEBUG trace level. For other levels, JDBC is a block call. This transaction type and context names are only available in application servers.

Transaction type: JDBC	
Context name	**Description**
ClassName	The interface name for the JDBC call. This name is not the actual class name. **Example:** java.sql.PreparedStatement
MethodName	The method name for the JDBC call. **Example:** executeUpdate()

Java Connector Architecture (JCA)

The JCA transaction type is available only at the DEBUG trace level. For other levels, JCA is a block call. This transaction type and context names are only available in application servers.

Transaction type: JCA	
Context name	**Description**
ClassName	The class name for the JCA call. **Example:** javax.resource.spi.ManagedConnection
MethodName	The method name for the JCA call. **Example:** getConnection(Subject, ConnectionRequestInfo)

Web Services Provider

The Web services provider transaction type is for server-side Web services requests. This transaction type and the *Wsdlport* and *Operation* context names are available in all types of servers. The *Transport*, *NameSpace*, and *InputMessage* context names are only available in application servers.

Transaction type: Web Services Provider	
Context name	**Description**
WsdlPort	The WSDL port name that is associated with the Web service. **Example:** AccountManager
Operation	The operation name that is associated with the Web service. **Example:** createNewAccount

Transaction type: Web Services Provider	
Transport	The transport name that is associated with the Web service. **Example:** http
NameSpace	The name space that is associated with the Web service. **Example:** http://accountmanager.mycorp.com
InputMessage	The input message name that is associated with the Web service. **Example:** createNewAccountRequest

Web Services Requestor

The Web services requestor transaction type is available only at DEBUG trace level. For other levels, Web services requestor is a block call. This transaction type and context names are only available in application servers.

Transaction type: Web Services Requestor	
Context name	**Description**
WsdlPort	The Web Services Description Language (WSDL) port name that is associated with the Web service. **Example:** AccountManager
Operation	The operation name that is associated with the Web service. **Example:** createNewAccount
Transport	The transport name that is associated with the Web service. **Example:** http
Parameters	The parameters for the Web service request. **Example:** customerName,addressStreet,addressCity,addressState,addressZip

Java Message Service (JMS)

The JMS transaction type is only for message-driven bean (MDB) scenarios in default messaging, including the System Integration Bus (SIB) layer and MQ. This transaction type and context names are only available in application servers.

Transaction type: JMS	
Context name	**Description**
DestinationName	The destination queue name or topic name for the Java Message Service (JMS). **Example:** MyBusiness.Topic.Space
MessageSelector	The message selector for JMS.
Provider	Provider refers to either default messaging, SIB, or MQ. **Example:** Default Messaging

JMS send and receive

The JMS send and receive transaction type is only available at the DEBUG trace level. For other levels, JMS send and receive is a block call. This transaction type and context names are only available in application servers.

Transaction type: JMS send/receive	
Context name	**Description**
ClassName	The class name in the JMS provider that makes the call. **Example:** com.ibm.ws.sib.api.jms.impl.JmsTopicPublisherImpl
MethodName	The method name in the JMS provider that makes the call. **Example:** sendMessage

SIB send and receive

The SIB send and receive transaction type is only available at the DEBUG trace level. For other levels, SIB send and receive is a block call. This transaction type and context names are only available in application servers.

Transaction type: SIB send/receive.	
Context name	**Description**
ClassName	The class name in the SIB layer that makes the call. **Example:** com.ibm.ws.sib.processor.impl.ProducerSessionImpl
MethodName	The method name in the SIB layer that makes the call. **Example:** send

SIB mediation

The SIB mediation transaction type and context names are only available in application servers.

Transaction type: SIB mediation	
Context name	**Description**
ClassName	The class name in which the call is made.
MethodName	The method name in which the call is made.
MediationName	The mediation name for the JMS. **Example:** myMediation
BusName	The service integration bus name for the JMS. **Example:** thisBusName
DestinationName	The destination name for the JMS. **Example:** myMessageQueue

AsyncBeans

The AsyncBeans transaction type is for AsyncBeans timers, alerts, and deferred starts. This transaction type and context names are only available in application servers.

Transaction type: AsyncBeans	
Context name	**Description**
Type	The type of AsyncBeans task. **Example:** COMMONJ_TIMER
ClassName	The class name that executes the AsyncBeans task. **Example:** com.mycorp.MyTaskClass

JNDI

The JNDI transaction type is only available at DEBUG trace level. For other trace levels, JNDI calls will not be instrumented. This transaction type and context names are only available in application servers.

Transaction type: JNDI	
Context name	**Description**
JNDIName	The JNDI lookup name. **Example:** ejbJndiName

Portlet

The portlet transaction type is and context names are only available in application servers.

Transaction type: Portlet	
Context name	**Description**
Method	The method of the portlet that is currently utilized, which can be either action or render.
WindowID	The window identifier of the portlet that is currently utilized.
URI	The context path of the current portlet request.

ARM correlators via HTTP and SOAP protocols

For HTTP inbound, WebSphere Application Server checks the case sensitive HTTP header “ARM_CORRELATOR” for the incoming ARM correlator. The application server does not allow the flow of ARM correlators via HTTP outbound.

For SOAP inbound, WebSphere Application Server checks the SOAP header for an element as follows:

- element =″arm_correlator″;
- namespace URI = “http://websphere.ibm.com”,
- prefix = ″reqmetrics″;
- actor URI = ″reqmetricsURI″;

For SOAP outbound, WebSphere Application Server creates a new SOAP header and passes the ARM correlator string as a text node in the header. For example:

- element =″arm_correlator″;
- namespace URI = “http://websphere.ibm.com”,
- prefix = ″reqmetrics″;
- actor URI = ″reqmetricsURI″;

The following is an example SOAP header with an ARM correlator. Note that the ARM correlator must be passed in hex string format of the ARM correlator byte array despite whether HTTP or SOAP protocol is used.

```
<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<reqmetrics:arm_correlator soapenv:actor="reqmetricsURI" xmlns:reqmetrics="http://websphere.ibm.com">
37000000000000000000
</reqmetrics:arm_correlator>
</soapenv:Header>
```

Isolating performance for specific types of requests

This topic describes how to enable request metrics filters.

Request metrics compares each incoming request to a set of known filters, but you need to enable these filters.

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request metrics** in the administrative console navigation tree.
3. Click **Filters**.
4. Click *filter type*.
5. Select the check box in the **Enable** field under the Configuration tab.
6. Click **Apply** or **OK**.
7. Click **Save**. You can enable or disable a filter group. If the group is enabled, you can enable or disable individual filters.

The request metrics filters are enabled according to your configuration. For example, if you enabled source IP, only requests whose source IP matches the one specified in the filter will be instrumented.

Note: Filters will only be checked for edge transactions. An edge transaction is the transaction that first enters an instrumented system. For example, if a servlet calls an Enterprise JavaBean, the servlet is the edge transaction, assuming it is not instrumented at the Web server plug-in, and the URI and SOURCE_IP filters will be checked for the servlet request. However, when the request comes to EJB container, EJB filter will not be checked because it is no longer an edge transaction.

If logging time is spent in the Web server, refer to “Regenerating the Web server plug-in configuration file” on page 2292.

Adding and removing request metrics filters

This topic summarizes how to add and remove request metrics filter.

To add or remove request metrics filters, perform the following steps:

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request metrics** in the console navigation tree.
3. Click **Filters**.
4. Choose a filter type.
   a. Click **Filter values**.

b. You can edit, add, and delete a filter value. To edit, click a filter value and change its value. To add, click **New** and type in the value and optionally check the **Enable filter** box. To delete, select a filter value and click **Delete**.

5. Click **Apply** or **OK**.
6. Click **Save**.

Adding or removing request metrics filters is complete.

If logging time spent in the Web server, refer to “Regenerating the Web server plug-in configuration file” on page 2292.

Request metrics filters

Use this page to view a list of request metrics filters.

To view this administrative console page, click **Monitoring and Tuning** > **Request metrics** > **Filters**.

Maximum rows: Specifies the maximum number of rows that displays when the collection is large. The rows that are not displayed appear on the next page.

Retain filter criteria: Specifies whether to use the same filter criteria entered in the show filter function to display this page the next time you visit it.

Type:

Specifies the type of request metrics filter.

Enable:

Specifies whether this filter is enabled. This option must be enabled to enable the filter values under this filter type.

Request metrics filter settings

Use this page to specify filters that define whether or not trace is enabled for inbound requests. Note that the filters will only be checked for inbound requests. For intra-process and outbound calls, filters will not be checked. If an inbound request passes the relevant filters, the request will be instrumented as it moves through WebSphere Application Server. If an inbound request does not pass the relevant filters, the request will not be instrumented for its entire code path in WebSphere Application Server.

To view this administrative console page, click **Monitoring and Tuning** > **Request metrics** > **Filters** > *filter_type*.

Type:

Specifies the type of request metrics filter.

Enable:

Specifies whether this filter is enabled.

Filter values:

Specifies the value(s) of the request metrics filter type and enablement for the value(s).

Filter values collection

Use this page to specify the values for source IP, URI, Web services, Java Message Service (JMS), or Enterprise JavaBeans (EJB) request metrics filters. When multiple filter values are enabled, a request will pass the filter as long as it matches one of the filter values.

To view this administrative console page, click **Monitoring and Tuning** > **Request metrics** > **Filters** > *filter_type* > **Filter values**.

Maximum rows: Specifies the maximum number of rows that displays when the collection is large. The rows that are not displayed appear on the next page.

Retain filter criteria: Specifies whether to use the same filter criteria entered in the show filter function to display this page the next time you visit it.

Value:

Specifies a source IP, URI, Web services, JMS, or EJB value based on the type of filter.

For example, for URI filters, the value might be ″/servlet/snoop″.

Enable filter:

Specifies whether a filter value is enabled.

Filter values settings

Use this page to specify the values for source IP, URI, Web services, Java Message Service (JMS), or Enterprise JavaBeans (EJB) 2method name request metrics filters.

To view this administrative console page, click **Monitoring and Tuning** > **Request metrics** > **Filters** > *filter_type* > **Filter values** > *filter_value*.

Value:

Specifies a source IP, URI, Web services, JMS, or EJB value based on the type of filter.

For example, for URI filters, the value can be `/servlet/snoop`. When needed, the wildcard (*) can be appended at the end of the value. For example, `/servlet/s*` is a valid URI filter value.

For JMS and Web services filter values, the wildcard can be used at the end of one or more name/value pairs.

Enable filter:

Specifies whether this filter value is enabled.

Specifying how much data to collect

This topic describes how to set the trace level to generate trace records in the administrative console.

Trace level specifies how much trace data to accumulate for a given transaction. You should choose the one of the following values when setting the trace level in the administrative console.

None No instrumentation.

Hops Generates instrumentation information on process boundaries only (for example, a servlet request coming from a browser or a Web server and a JDBC request going to a database).

Performance_debug
Generates the data at Hops level and the first level of the intra-process servlet and Enterprise JavaBeans (EJB) call (for example, when an inbound servlet forwards to a servlet and an inbound EJB calls another EJB). Other intra-process calls like naming and service integration bus (SIB) are not enabled at this level.

Debug
Provides detailed instrumentation data, including response times for all intra-process calls.

Note: Requests to servlet filters will only be instrumented at this level.

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request metrics** in the administrative console navigation tree.
3. Find **Trace level** in the Configuration tab.
4. Select a trace level from the drop down list box. To set the request metrics trace level to generate records, make sure that the trace level is set to a value greater than `None`.
5. Click **Apply** or **OK**.
6. Click **Save**.

Information will be generated to reflect the trace level you selected.

Regenerate the Web server plug-in configuration file as described in the “Regenerating the Web server plug-in configuration file” on page 2292 file, if logging time is spent in the Web server.

Request metrics trace filters

When request metrics is active, trace filters control which requests get traced. The data is recorded to the system log file or sent through Application Response Measurement (ARM) for real-time and historical analysis.

Incoming HTTP requests

HTTP requests that arrive at WebSphere Application Server might be filtered based on the URI or the IP address or both of the originator of the request.

.

- **Source IP address filters.** Requests are filtered based on a known IP address. You can specify a mask for an IP address using the asterisk (*). If used, the asterisk must always be the last character of the mask, for example 127.0.0.*, 127.0.*, 127*. For performance reasons, the pattern matches character by character, until either an asterisk is found in the filter, a mismatch occurs, or the filters are found to be an exact match.

  Only addresses that are entered in one of the following addressing formats are acceptable as the source IP addresses:
  - The IPv4 addressing format:

    ```
    ^(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])$
    ```
  - The IPv6 addressing format:

    ```
    ^([0-9a-fA-F]*[0-9a-fA-F]*[0-9a-fA-F]*[0-9a-fA-F]*:){1,7}([0-9a-fA-F]*
    [0-9a-fA-F]*[0-9a-fA-F]*[0-9a-fA-F]*)$
    ```
  - The IPv4 compatiable IPv4 addressing format:

    ```
    ^([0]*[0]*[0]*[0]*:){1,7}((\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5]))$/<constant-value>
    ```
  - The IPv4MappedIPv6 addressing format:

    ```
    ^([0]*[0]*[0]*[0]*:){1,6}([fF][fF][fF][fF]:){1}((\d{1,2}|1\d\d|2[0-4]
    \d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5]))$
    ```

Note: If the client machine is a dual stack machine and its IP address is specified as the source IP address that is filtered by the request metrics filter, you must specify the IPv4 addressing format rather than the IPv6 addressing format. Only if the client machine is a single stack IPv6 machine, can you specify it as the IPv6 addressing format

- **URI filters.** Requests are filtered, based on the URI of the incoming HTTP requests. The rules for pattern matching are the same as for matching source IP address filters.
- **Filter combinations.** If both URI and source IP address filters are active, request metrics requires a match for both filter types. If neither is active, all requests are considered a match.

Incoming enterprise bean requests

Incoming enterprise bean requests

- Requests are filtered based on the full name of the EJB method. As with IP address and URI filters, the asterisk (*) might be used in the mask. If used, the asterisk must always be the last character of a filter pattern.

Because the ability to track the request response times comes with a cost, filtering helps optimize performance when using request metrics.

The Web services filter and the Java Message Service (JMS) filter was added to the WebSphere Application Server, Version 6 product. The filter values for Web services are a combination of a Web Services Description Language (WSDL) port name, operation name, and transport name. The filter value for JMS is the destination name.

Related tasks

"Specifying how much data to collect" on page 2290
This topic describes how to set the trace level to generate trace records in the administrative console.

Regenerating the Web server plug-in configuration file

This topic describes the steps to regenerate the Web server plug-in configuration file after you modify the request metrics configuration.

After modifying the request metrics configuration, you must complete the following steps to regenerate the Web server plug-in configuration file. Regeneration ensures that the Web server plug-in recognizes the changes that you made for the request metrics configuration. If you make multiple changes to request metrics, then regenerate the plug-in configuration files when you complete all the changes.

Important: You must complete this step after you change the request metrics configuration. If you do not, the Web server plug-in might have different request metrics configuration data than the application server. This difference in configuration data might cause inconsistent behaviors for request metrics between the Web server plug-in and the application server.

1. Open the administrative console.
2. Click **Server > Web servers** .
3. Select the *Web server* check box.
4. Click **Generate Plug-in**.

The regeneration of the Web server plug-in configuration file is complete.

Enabling and disabling logging

This topic describes how to enable and disable logging through the administrative console.

You can enable and disable logging through the administrative console to start the generation of request metrics logs in the SystemOut.log file in the following directory:

profile_root/logs/*server_name*

1. Open the administrative console.

2. Click **Monitoring and Tuning > Request metrics**.
3. Under **Request Metrics Destination**, select the **Standard Logs** check box to enable the logging feature. To disable the logging , clear the **Standard Logs** check box.

The logging feature is enabled. You can view the generated request metrics logs in the SystemOut.log file.

Request metrics performance data

Use this page to learn how to interpret performance data for request metrics in trace record format.

The trace records for request metrics data are output to two log files: the Web server plug-in log file and the application server log file. The default directory for these log files is *plugin_install_root*/logs/*web_server_name*/http_plugin.log and *install_root*/profiles/*profile_name*/logs/*server_name*) and the default names are SystemOut.log and http_plugin.log. You might, however, specify these log file names and their locations.

In the WebSphere Application Server log file the trace record format is:

```
PMRM0003I: parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
-
current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
           type=TTT detail=some_detail_information elapsed=nnnn
```

In the Web server plug-in log file the trace record format is:

```
PLUGIN:
parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
- current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
           type=TTT detail=some_detail_information elapsed=nnnn bytesIn=nnnn
           bytesOut=nnnn
```

The trace record format is composed of two correlators: a parent correlator and current correlator. The parent correlator represents the upstream request and the current correlator represents the current operation. If the parent and current correlators are the same, then the record represents an operation that occurs as it enters WebSphere Application Server.

To correlate trace records for a particular request, collect records with a message ID of PMRM0003I from the appropriate application server log files and the PLUGIN trace record from the Web server plug-in log file. Records are correlated by matching current correlators to parent correlators. You can create the logical tree by connecting the current correlators of parent trace records to the parent correlators of child records. This tree shows the progression of the request across the server cluster. Refer to “Why use request metrics?” on page 2275 for an example of the transaction flow.

The parent correlator is denoted by the comma separating fields following the keyword, parent:. Likewise, the current correlator is denoted by the comma separating fields following, current:.

The fields of both parent and current correlators are:

- **ver:** The version of the correlator. For convenience, it is duplicated in both the parent and current correlators.
- **ip:** The IP address of the node of the application server that generated the correlator. If the system has multiple IP addresses, request metrics uses one of the IP addresses to identify the system.
- **pid:** The process ID of the application server that generated the correlator.
- **time:** The start time of the application server process that generated the correlator.
- **reqid:** An ID that is assigned to the request by request metrics, unique to the application server process.
- **event:** An event ID that is assigned to differentiate the actual trace events.

Following the parent and current correlators, the metrics data for timed operation are:

- **type**: A code that is represents the type of operation being timed. Supported types include `HTTP`, `URI`, `EJB`, `JDBC`, `JMS`, `COMMONJ_WORK_POOLED`, `COMMONJ_TIMER`, `Web services requester`, and `Web services provider`.
- **detail**: Identifies the name of the operation being timed (See the following description of Universal Resource Identifier (URI), `HTTP`, `EJB`, `JDBC`, `JMS`, `asynchronous beans`, and `Web services`.)
- **elapsed**: The measured elapsed time in <units> for this operation, which includes all sub-operations called by this operation. The unit of elapsed time is milliseconds.
- **`bytesIn`**: The number of bytes from the request that is received by the Web server plug-in.
- **`bytesOut`**: The number of bytes from the reply that is sent from the Web server plug-in to the client.

The type and detail fields that are described include:
- **`HTTP`**: The Web server plug-in generates the trace record. The detail is the name of the URI that is used to invoke the request.
- **`URI`**: The trace record is generated by a Web component. The URI is the name of the URI that is used to invoke the request.
- **`EJB`**: The fully qualified package and the method name of the enterprise bean.
- **`JDBC`** The interface name and method name for that JDBC call.
- **`JMS:`** JMS includes the particulars of various JMS parameters
- **`Asynchronous beans:`**The detail specifies the name of the asynchronous beans. Asynchronous beans include two types: `COMMONJ_WORK_POOLED` and `COMMONJ_TIMER`.
- **`Web services:`**Web services include the particulars of various Web services parameters. Web services include two types: `Web services requestor` and `Web services provider`.
- **SIB**: Used for instrumentation in service integration bus including message send/receive and mediation.
- **JCA**: J2EE Connector Architecture. The detail specifies the class name in which the JCA call is made.
- **JNDI**: Used for JNDI naming look up. The detail specifies the JNDI name.
- **`JMS send and receive:`** Generates the trace record by JMS sending and receiving messages.
- **`SIB send and receive:`** Generates the trace record by SIB sending and receiving messages.

Related tasks

"Enabling and disabling logging" on page 2292
This topic describes how to enable and disable logging through the administrative console.

Request metric extension

Certain applications might require additional instrumentation points within the request metrics flow. For example, you might want to understand the response time to a unique back-end system as seen in the following call graph:

```
HTTP request /trade/scenario             ------------------------------> 172 ms
     Servlet/trade/scenario          -------------------------------> 130 ms
       Servlet/call to unique back-end system -------------------------->38 ms
```

Request metrics uses a *token* or *correlator* when tracing the flow of each request through the system. To create the call graph above with this instrumentation, you must plug into that flow of the request and issue the appropriate Application Response Measurement (ARM) API for an ARM agent to collect the data and for the ARM vendor to create the call graph.

Request metrics exposes the Correlation Service API for you to plug into the flow of the request. The following example is one of the typical flows that might be followed by an instrumented application to plug into the request metrics flow:

1. Create a new ArmTransaction object, which runs various instrumentation calls such as start or stop. The Correlation Service Arm wrapper (PmiRmArmTx) encapsulates this object before being inserted into the request metrics flow.
2. Populate the ArmTransaction object with an appropriate ArmCorrelator object. This object encapsulates the actual ARM correlator bytes.
3. Run the start method on the ArmTransaction object, marking the beginning of the instrumented method.

4. Instantiate a PmiRmArmTx object using the static method on the PmiRmArmTxFactory class, and populate it with the ArmTransaction object above.
5. Pass the PmiRmArmTx object above to the Correlation Service by pushing it onto the Correlation Service stack using exposed methods on the PmiRmArmStack class.
6. Perform the tasks that need to be done by the method being instrumented. The Correlation Service takes care of flowing the ArmTransaction object as necessary, which eventually results in the call graph view of the transaction times.
7. At the end of the instrumented method, access the PmiRmArmTx object from the Correlation Service using exposed methods on the PmiRmArmStack class, access the ArmTransaction object and perform a stop to indicate the end of the transaction.

Example: Using the correlation service interface

Consult this information when utilizing the ARM API with the correlation service as part of a servlet instrumentation.

The arm40 binaries should be installed in accordance with the installation instructions supplied by the implementation provider. Once this is done, restart the server. This causes trace records to be generated in the SystemOut.log file indicating the instantiation of the appropriate ARM implementation.

The following example illustrates one of the typical workflows of using the ARM API in conjunction with the correlation service as part of a servlet instrumentation:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

PmiRmArmTx artrax =
// The factory detects the currently active ARM implementation (specified by user through
// admin console) and instantiates an appropriate ARM wrapper object

    PmiRmArmTxFactory.createPmiRmArmTx();
ArmTransaction at = newArmTx();
if (null == at)
            out.println("Got a null ArmTransaction");
ArmCorrelator arc = newArmCorr();
at.start(arc);
try {
 artrax.setArmTransaction(at);
 PmiRmArmStack.pushTransaction(artrax);
} catch (Exception e) {
 System.out.println("Caught 1 exception" + e);
}

PmiRmArmTx atxwrp = PmiRmArmStack.peekTransaction();

if (atxwrp == null)
 out.println("Armtransaction is null");

//getArmType
try {
 out.println("ARMTYPE is"+ PmiRmArmTx.getARMType());
} catch (Exception e) {
 out.println(e);
}
//getting correlator bytes
try {
 if (null == atxwrp.getCorrelatorBytes())
  out.println("Got a null Correlator");
} catch (Exception e) {
 out.println(e);
}

//blocked/unblocked
```

```
long blkid = 0;
try {
 out.println(blkid = atxwrp.blocked());
} catch (Exception e) {
 out.println(e);
}

try {
 out.println(atxwrp.unblocked(blkid));
} catch (Exception e) {
 out.println(e);
}
try {
 atxwrp = PmiRmArmStack.popTransaction();
 ArmTransaction art = (ArmTransaction) atxwrp.getArmTransaction();
 art.stop(ArmConstants.STATUS_GOOD);
} catch (Exception e) {
 out.println(e);
}

}



private ArmTransaction newArmTx() {

 ArmTransactionFactory txFactory = null;
try {
 String sWasName = "WebSphere";
 String appName = "t23xpimage/t23xpimage/server1";
 String sCellName = appName.substring(0, appName.indexOf("/"));
 String sNodeInstance =
   appName.substring(appName.indexOf("/") + 1, appName.length());
 sNodeInstance = sNodeInstance.replace('/', '.');
 txFactory = (ArmTransactionFactory)
 newObjectInstance("org.opengroup.arm40.sdk.ArmTransactionFactoryImpl");
 ArmApplication app = null; // 149297
 ArmApplicationDefinition appDef = null; //LIDB3207
 appDef =    txFactory.newArmApplicationDefinition(sWasName, null, null);
 app = txFactory.newArmApplication(appDef, sCellName, sNodeInstance, null);

 String[] idnames = { "request_type" };
 String[] idvalues = { "URI" };
 String[] ctxnames = { "URI" };
 ArmIdentityPropertiesTransaction props =
   txFactory.newArmIdentityPropertiesTransaction(
     idnames,
     idvalues,
     ctxnames,
     null);
 ArmTransactionDefinition atd =
    txFactory.newArmTransactionDefinition(
     appDef,
     "URI",
     props,
     (ArmID) null);
 ArmTransaction at = txFactory.newArmTransaction(app, atd);
 return at;
} catch (Exception e) {
 System.out.println(e);
 return null;
}

}

private ArmCorrelator newArmCorr() {
```

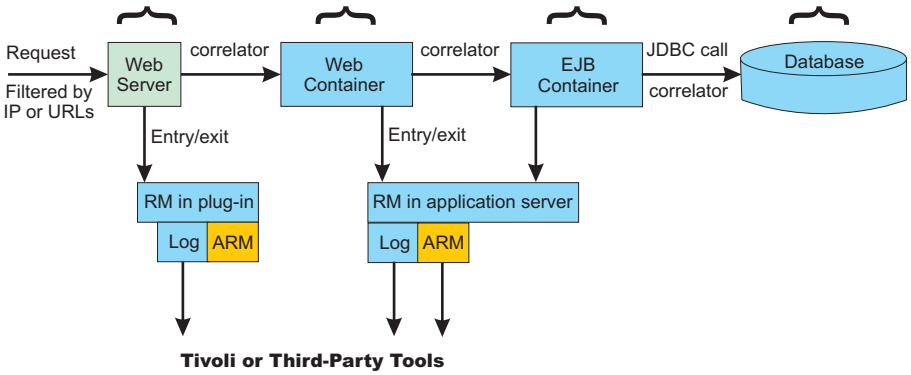
```
 ArmTransactionFactory txFactory = null;
try {
 String sWasName = "WebSphere";
 String appName = "t23xpimage/t23xpimage/server1";
 txFactory =
 (ArmTransactionFactory) newObjectInstance("org.opengroup.arm40.sdk.ArmTransactionFactoryImpl");

 ArmCorrelator arc =txFactory.newArmCorrelator(
     PmiRmArmStack.peekTransaction().getCorrelatorBytes());
 return arc;
} catch (Exception e) {
 System.out.println(e);
 return null;
}

}
```

There are several potential scenarios for using the PmiRmArmStack. This example shows a scenario where code accesses an existing PmiRmArmTx on the stack, extracts the correlator, and calls blocked and unblocked. This is a typical scenario when sending a correlator along an unsupported protocol. In this scenario, the Arm transaction is already on the stack.

```
1   PmiRmArmTx artrax =
2    PmiRmArmStack.peekTransaction();
3   if( artrax != null )
4                         {
5                                    try
6                                    {
7        byte[] cbytes = artrax.getCorrelatorBytes();
8                                       stuffBytesIntoOutboundMessage( msg, cbytes);
9                                       long blockedId = 0;
10                                      try
11                                      {
12                                         blockedId = artrax.blocked();
13                                      }
14                                      catch( NoSuchMethodException nsme )
15                                      {
16                                          // must not be running ARM4 or eWLM
17                                      }
18                                         sendMsg( msg );
19                                      try
20                                      {
21                                         artrax.blocked( blockedId );
22                                      }
23                                      catch( NoSuchMethodException nsme )
24                                      {
25                                          // must not be running ARM4 or eWLM
26                                      }
27
28                                   }
29                                   catch( Exception e )
30                                   {
31                                      report a problem;
32                                   }
33  }
```

Differences between Performance Monitoring Infrastructure and request metrics

Performance Monitoring Infrastructure (PMI) provides information about average system resource usage statistics, with no correlation between the data across different WebSphere Application Server components. For example, PMI provides information about average thread pool usage. Request metrics provides data about each individual transaction, correlating this information across the various WebSphere Application Server components to provide an end-to-end picture of the transaction, as shown in the following diagram:



Chapter 26. Troubleshooting deployment

- Select the problem you are having with deploying or installing developed code for WebSphere Application Server.
  - Errors or problems deploying, installing, or promoting applications
  - Class loader exceptions
- To troubleshoot other deployment issues, use the following resources.
  - 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 IBM Support page.
  - 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.

Errors or problems deploying, installing, or promoting applications

This topic describes problems that you might encounter when deploying, installing, or promoting applications and suggests ways to resolve the problems.

What kind of problem are you having?

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 by looking at available online support including hints and tips, technotes, and fixes. If the problem has not been identified, see Troubleshooting help from IBM.

I installed my application using the wsadmin tool, but the application does not display under Applications > 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.

On the SuSE9 or other Linux platform, you can either increase the number of files that can be opened to resolve the problem or you can modify the application to close files with disciplines. To increase the number of files that you can open at the same time, run the following command in the shell before invoking the process that needs to open a number of files:

```
ulimit -n number_of_files
```

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.

I get a `java.lang.RuntimeException: Failed_saving_bytes_to_wor_ERROR_` error in the assembly tool, administrative console or the wsadmin tool.

If you see this error when attempting to generate deployed code in an assembly tool, installing an application or module in the administrative console, or using the wsadmin tool to install an application or module, the file path length of the temporary system file might be exceeded. This situation is typically an issue only on Windows platforms.

To verify this problem, check the `TEMP` and `TMP` environment variables for your system. Long environment variables add path length to the file names accessed by the EJB deployment tool.

To resolve the problem:
1. Stop all WebSphere Application Server processes and close all DOS prompts.
2. Set the `TMP` and `TEMP` environment variables to something short, for example `C:\TMP` and `C:\TEMP`.
3. Reinstall the application.

Otherwise, try rebooting and redeploying or reinstalling the application.

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:

```
WASX7015E: Exception running command: "$AdminApp installInteractive C:/Documents and Settings/
myUserName/Desktop/MyApp/myapp.ear"; exception information:
com.ibm.bsf.BSFException: error while
eval'ing 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:

```
$AdminApp installInteractive "C:\Documents
and Settings\myUserName\Desktop\MyApps\myapp.ear"
```

- If the full text of the error is similar to:

```
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\MyAppsmyapp.ear"
```

  The application path is incorrectly specified. In this case, you must use ″forward-slash″ (/) separators in the path.

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 at http://www.ibm.com/cgi-bin/db2www/data/db2 /udb/winos2unix/support/index.d2w/report.

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:

- For operating systems other than Linux, edit the DDL in the vi editor, removing the Ctl-M character at the beginning of each line.
- For Linux systems, regenerate the deployment code for the application EAR file on a Linux platform.

Error message ADMA0004E: Validation error in task Specifying the Default Datasource for EJB Modules returned when installing application using the administrative console or the wsadmin tool

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 beannameBean 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 versions 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. For details on the syntax, see Installing applications with the wsadmin tool.

Cannot load resource WEB-INF/ibm-web-bnd.xmi 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 server. The application fails to install because the WEB-INF/ibm-web-bnd.xmi file contains xmi tags that the underlying WCCM model no longer recognizes.

The following error messages display:

```
IWAE0007E Could not load resource "WEB-INF/ibm-web-bnd.xmi" 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.xmi" 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.xmi" in archive "tmp.war"
!Stack_trace_of_nested_exce!
com.ibm.etools.j2ee.exception.WrappedRuntimeException: Exception occurred loading
WEB-INF/ibm-web-bnd.xmi
!Stack_trace_of_nested_exce!
```

To work around this problem, remove the xmi:type=EJBLocalRef tag from the ibm-web-bnd.xmi 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 v5.1, v6.0, and later releases.

Error message 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.

″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.

I get a NameNotFoundException message when deploying an application that contains an EJB module

If you specify that EJB deploy 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.

During application installation, the call to EJB deploy throws an exception

When you specify that the EJB deployment tool be run during application installation and if installation fails with the error command line too long, the problem is that the deployment command generated during installation exceeds the character limit for a command line on the Windows platform. This problem occurs only on Windows platforms.

To work around this problem, you can reduce the length of the EAR file name, reduce the length of the JAR file name within the EAR file, reduce the class path or other options specified for deployment, or change the %TEMP% location of the Windows system to make its path shorter.

I get 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 product, it is compiled using the Version 6.x run-time 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.

Troubleshooting testing and first time run problems

Select the problem you are having with testing or the first run of deployed code for WebSphere Application Server:
- “The server process does not start or starts with errors” on page 2323.
- “The application does not start or starts with errors” on page 2308.
- “A Web resource does not display” on page 2310.
- “Cannot access a data source” on page 775.
- “Cannot access an enterprise bean from a servlet, a JSP file, a stand-alone program, or another client” on page 200.

- Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client.
- Access problems after enabling security.
- Errors after enabling security.
- Errors after configuring or enabling Secure Sockets Layer.
- Errors in messaging.
- .
- A client program does not work.
- Errors connecting to WebSphere MQ and creating WebSphere MQ queue connection factory.

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 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 page.

Errors starting an application

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 2305
- "Graphics do not appear in the JSP file or servlet output" on page 2305
- "SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file" on page 2306
- "After modifying and saving a JSP file, the change does not show up in the browser (the old JSP file displays)" on page 2307
- "Message like ″Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing″ appears when attempting to browse JSP file" on page 2307
- "The Java source generated from a JSP file is not retained in the temp directory (only the class file is found)" on page 2307
- "The JSP Batch Compiler fails with the message ″Enterprise Application [application name you typed in] not found.″" on page 2308
- "There is a translation problem with non-English browser input" on page 2308
- "Scroll bars do not appear around items in the browser window" on page 2308
- "Error ″Page cannot be displayed... server not found or DNS error″ appears when attempting to browse a JavaServer Pages (JSP) file using Internet Explorer" on page 2308

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.
   - Retry the Web request.
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.

  Typically this directory will be found in the *install_root*/installedApps/*nodename*/*appname*.ear directory or *install_root*/installedApps/*nodename*/*appname*Network.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.

     The file typically is found in the *install_root*/config/cells/*nodename* or *nodename*Network/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. WebSphere Application Server Version 5 follows the J2EE standard, which means that the document root is the *Web_module_name*.war directory of your deployed Web application.

  Typically, this directory is found in the *install_root*/installedApps/*nodename*/*appname*.ear directory or *install_root*/installedApps/*nodename*/*appname*Network.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: **<img SRC="images/banner.gif">**, not <img SRC="banner.gif">.

- 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.

     The file typically is found in the *install_root*/config/cells/*nodename* or *nodename*Network/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.

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;
      ^
```

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`.

  For example, to access myCom.myServlet.class, located in c:\WebSphere\ApplicationServer\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`)

After modifying and saving a JSP file, the change does not show up in the browser (the old JSP file displays)

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.

Message like "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.

The Java source generated from a JSP file is not retained in the temp directory (only the class file is found)

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.

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.

The directory path is *install_root*`\config\cells\`*node_name*`Network\applications`.

For example:
- ″`JspBatchCompiler -enterpriseapp.name sampleApp.ear/deployments/sampleApp`″ is correct, as opposed to
- ″`JspBatchCompiler -enterpriseapp.name sampleApp`″, which is incorrect.

There is a 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″ 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**.

The application does not start or starts with errors

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?
- A “java.lang.ClassNotFoundException: classname Bean_AdderServiceHome_04f0e027Bean” on page 2309 error occurs
- A “ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter” on page 2309 error occurs

- "NMSV0605E: ″A Reference object looked up from the context...″ error when starting an application" on page 2310.
- A parsing error when running an application that uses the JSF configuration occurs.

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: *install_dir*/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 the ″Messages″ heading.

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.

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.

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 such as **WebSphere Application Server v6**. If you do not have access to the target server, you can specify a false location such as c:\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 standalone 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 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's 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 UrlContextHel 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 that the JNDI Name given in administrative console under **Resources-> Manage JDBC Provider > 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, or
3. 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.

Parsing error when running an application that uses the JSF configuration

If you are using double-byte characters in the profile name, you receive a parsing error when running an application that uses the JavaServer Faces™ (JSF) configuration. The problem is related to the JSF configuration that is part of the `jsf-ibm.jar`, which is included when building JSF applications in Rational® Application Development (RAD). The configuration files are referencing entities from inside the main `faces-config.xml` file.

Avoid using double-byte characters when you create a profile.

A Web resource does not display

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, the Application Server might not be serving the target resource. Try accessing 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 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` in the WebSphere Application Server directory structure) to your Web module document root, and try to access it. If successful, the problem is with your resource.

      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 some kinds of 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*.

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.

On Windows systems, look in the Services panel for the service corresponding to your HTTP server, and verify that the state is **Started**. If not, start it. If the service does not start, try starting it manually from the command prompt. If you are using IBM HTTP Server, the command is *IHS_install_dir*\apache .

On UNIX systems, execute the ps -ef | grep httpd command. There should be several processes running with a name of "httpd". If not, start your HTTP server manually. If you are using IBM HTTP Server, the command is *IHS_install_dir*/bin/apachectl start.

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

Starting with WebSphere Application Server Version 4.0, you can bypass the HTTP server and access a Web resource through the application server. It is not recommended to serve a production Web site 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 WebSphere administrative console, click **Servers>Manage Application Servers**.
   b. Select the target server, then under Additional Properties click **Web Container**.
   c. Under the Additional Properties of the Web container, click **HTTP Transports**. You see the ports listed for virtual hosts served by the application server.
   d. 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 **Application Servers > Your_ApplicationServer > Web Container > HTTP Transports** to check the Default virtual host and the HTTP transport ports used by this application server.
   b. Click **Environment > Manage 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.

Cannot uninstall an application or remove a node or application server

What kind of problem are you having?
- After uninstalling an application through wsadmin tool, the application continues to run and throws ″DocumentIOException″

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.

Chapter 27. Troubleshooting administration

Use this information if you are having problems with administrative functions.

- Select the problem you are experiencing.
  - I have problems bringing up or using the administrative console.
  - I have problems starting or using the **wsadmin** command prompt.
  - I have problems using command line tools.
  - My Web module or application server dies or hangs.
  - I get errors trying to configure and enable security.
  - I cannot uninstall or remove a node or application server.
  - I have problems creating or using HTTP sessions.
  - I have problems using tracing, logging, log files, or other troubleshooting features.
  - I get errors connecting to the administrative console from a browser.
  - The stopServer.sh (base and ND) hangs and creates a Java core dump (Red Hat Linux).
- If you did not solve the problem, prepare to contact IBM support.

  If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

  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 IBM Support page.

Administration and administrative console troubleshooting

In WebSphere Application Server products, administrative functions are supported by:
- The application server (such as server1) process

The process must be running to use the administrative console. The **wsadmin** command-line utility has a local mode that you can use to perform some administrative functions, even when the server process is not running.

What kind of problem are you seeing?
- “Server status and messages in the console view are not current”
- “Role-based authorization fails” on page 2316
- “When starting or stopping a server using a wsadmin interactive scripting session, you receive an exception indicating read timed out” on page 2317
- “Problems starting or using the administrative console or wsadmin utility” on page 2317

Server status and messages in the console view are not current

When connecting to an Application Server that uses a Simple Object Access Protocol (SOAP) connection for a long time, the following problems begin to occur:

- Under the status column in the Servers view on an administrative console panel, the status of the server does not refresh.
- Server messages are not updated in the administrative console.
- A decrease of system resources occurs as numerous ports are created and left in the TIME_WAIT state.

This problem persists even after you restart the server or you start another server that uses the SOAP connection port.

The problem occurs because the SOAP connector does not support connection pooling. If the Application Server has many ongoing operations that use the SOAP connector, the Application Server quickly opens

and closes many ports. Due to the nature of the underlying TCP/IP protocol, these ports remain in the TIME_WAIT state for some time before the operating system can reclaim them. The number of ports that WebSphere Application Server opens can exceed the limit that the operating system imposes. Under this condition, the opening of additional ports fails through the SOAP connector until the operating system reclaims ports.

Use the following options to work around the problem:

- Increase the operating system limits on the number of ports.
- For WebSphere Application Server Toolkit, the wsadmin utility, or Java applications that use the Java Management Extension (JMX) connectors, switch to the Remote Method Invocation (RMI) connector.
- Wait until few or no ports are in the TIME_WAIT state before performing new operations through WebSphere Application Server Toolkit or the administrative console.

Role-based authorization fails

When you make a Java Management Extension (JMX) call such as getAttribute, setAttribute, invoke, and so on in your application, the caller requires an administrative role with sufficient permissions. The required role depends on the MBean attribute or method that the JMX caller calls and can be one of administrator, configurator, monitor, or operator. If one of the administrative roles is not assigned to the caller, or if the role is assigned, but the caller does not have the required permissions, the application receives a role-based authorization failure, for example:

```
SECJ0305I: Role based authorization check failed for securityname server.domain.name:3890/user.id,
accessId user:server.domain.name:3890/uid=user.id,ou=xxxx,dc=yyy,dc=zzz while invoking method
getNodeName on resource Server and module Server.
```

If the caller of the application cannot be assigned one of the administrative roles, the application can log in with one of the roles on behalf of the caller. For example:

```
try
{
 // Create a LoginContext to authenticate a user ID and password.
   javax.security.auth.login.LoginContext
   lc = new javax.security.auth.login.LoginContext("WSLogin",
  new com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl("adminuser",
   "adminpassword"));

 // perform the login
 lc.login();

 // Get the authenticated subject.
 javax.security.auth.Subject adminSubject = lc.getSubject();

 // Define the action that will take place using the authenticated Subject
 // You can define this action anywhere in the code, the action
  // is reference in the WSSubject.doAs that follows.
 java.security.PrivilegedAction adminAction = new java.security.PrivilegedAction()
 {
  public Object run()
  {
   try
   {
    // Get the WebSphere AdminService.
    AdminService adminservice = AdminServiceFactory.getAdminService();

    // Get the WebSphere Admin Local Server MBean instance.
    ObjectName objectname = adminservice.getLocalServer();

    // Get the Node name.
    String nodeName = (String)adminservice.getAttribute(objectname, "nodeName");

    // Get the Application Server name.
    String serverName = (String)adminservice.getAttribute(objectname, "name");
```

```
    // Get the Application Server Process ID.
    String serverPid = (String)adminservice.getAttribute(objectname, "pid");

    // Return a result, for this example, just return the serverPid.
    return serverPid;
   }
   catch (Exception e)
   {
    e.printStackTrace();
   }
   return null;
  }
 });

 // Invoke an AdminClient resource using the authenticated subject.
    // This example demonstrates the action of creating an
    // administrative client and returning a String value to use outside
    // the doAs block.
 String myData = (String)
    com.ibm.websphere.security.auth.WSSubject.doAs(adminSubject, adminAction);

 // use "myData" later on....
}
catch (javax.security.auth.login.LoginException e)
{
 e.printStackTrace();
}
```

When starting or stopping a server using a wsadmin interactive scripting session, you receive an exception indicating `read timed out`

When starting or stopping a server using a wsadmin interactive scripting session, you receive an exception indicating `read timed out`, for example:

```
WASX7015E: Exception running command: "$AdminControl startServer server1 Node1";
exception information:  com.ibm.websphere.management.exception.ConnectorException
org.apache.soap.SOAPException: [SOAPException: faultCode=SOAP-ENV:Client; msg=Read
timed out; targetException=java.net.SocketTimeoutException: Read timed out]
```

This exception occurs because the timeout value is too small. Increase the timeout value specified by the com.ibm.SOAP.requestTimeout property in the `soap.client.props` file in the *app_server_root*`/profiles/`*profile_name*`/properties` directory for a single server edition. The value you choose depends on a number of factors such as the size and the number of the applications installed on the server, the speed of your machine, and the level of usage of your machine. The default value of the com.ibm.SOAP.requestTimeout property is 180 seconds.

Problems starting or using the administrative console or wsadmin utility

If you have problems starting or using the administrative console or wsadmin utility, verify that the supporting server process is started and that it is healthy.

- For the application server process, look at these files:
  - *app_server_root*`/profiles/`*profile_name*`/logs/`*server_name*`/startServer.log` for the message that indicates that the server started successfully: `ADMU3000I: Server server1 open for e-business; process id is nnnn.`.
  - *app_server_root*`/profiles/`*profile_name*`/logs/`*server_name*`/SystemOut.log`
- Look up any error messages in these files in the message reference table. Select the **Reference** view in the information center navigation, and click **Messages**. A message like `WASX7213I: This scripting client is not connected to a server process` when trying to start wsadmin indicates that either the server process is not running, the host machine where it is running is not accessible, or that the port or server name that the wsadmin utility uses is incorrect.

- Verify that you are using the right port number to communicate with the administrative console or the wsadmin server:
  - Look in the `SystemOut.log` file.
    - The line `ADMC0013I: SOAP connector available at port` *nnnn* indicates the port that the server is using to listen for wsadmin functions.
    - The com.ibm.ws.scripting.port property in the *app_server_root*`/profiles/`*profile_name*`/properties/wsadmin.properties` file controls the port used by the wsadmin utility to send requests to the server.
  - If port value is different from the value shown in the `SystemOut.log` file, either change the port number in the `wsadmin.properties` file, or specify the correct port number when starting the wsadmin utility by using the -port *port_number* property on the command line.

    The com.ibm.ws.scripting.port property in the *app_server_root*`/profiles/`*profile_name*`/properties/wsadmin.properties` file controls the port used by the wsadmin utility to send requests to the server.
  - If the port value is different than the one specified in the Web address for the administrative console, change the Web address in the browser to the correct value. The default value is http://localhost:9060/ibm/console.
- Use the **telnet** command to test that the host name where the application server is running, is reachable from the system where the browser or wsadmin program is used. If you can ping the host name, no firewall or connectivity issues exist.
- If the host where the application server is running is remote to the machine from which the client browser or wsadmin command is running, ensure that the appropriate host name parameter is correct. Verify:
  - The host name in the browser Web address for the console.
  - The -host *host name* option of the **wsadmin** command that is used to direct the wsadmin utility to the right server
- Tracing the administrative component: WebSphere Application Server technical support might ask you to trace the administrative component for detailed problem determination. The trace specification for this component is com.ibm.websphere.management.*=all=enabled:com.ibm.ws.management.*=all=enabled″

If none of these steps solves the problem, see if the specific problem you are having is addressed in the Installation completes but the administrative console does not start topic. Check to see if the problem has been identified and documented using the links in the Diagnosing and fixing problems: Resources for learning topic. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

For current information available from IBM Support on known problems and their resolution, see the following topics on the IBM support page:

- Administrative Console
- Administrative Scripting Tools
- System management

IBM Support has documents that can save you time gathering the information that is needed to resolve this problem. Before opening a PMR, see the following topics on information gathering on the IBM support page:

- Administrative Console
- Administrative Scripting Tools
- System management

Installation completes but the administrative console does not start

This topic discusses problems that you can encounter when you attempt to access the console.

What kind of problem are you having?

- “An Internal Server Error, Page cannot be found, 404, or similar error occurs trying to view the administrative console”
- “An Unable to process login. Check user ID and password and try again. error occurs when trying to access the administrative console page”
- “The directory paths in the administrative console contain strange characters”

If you can bring up the browser page, but the administrative console behavior is inconsistent, error prone, or unresponsive, try upgrading your browser. Older browsers might not support all the features of the administrative console.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in Troubleshooting help from IBM. Before opening a problem report, see the Support page:

- http://www.ibm.com/software/webservers/appserv/was/support/

An Internal Server Error, Page cannot be found, 404, or similar error occurs trying to view the administrative console

Here are some steps to try if you are unable to view the administrative console:

- Verify that the application server that supports the administrative console is up and running. For a base configuration, the administrative console is deployed by default on server1.

  Before viewing the administrative console, you must take one of the following actions:
  - Run the **startServer server1** command for the Windows platform from a command prompt in the *install_dir*\bindirectory, or the **./startServer.sh server1** command for operating systems such as AIX or Linux.
  - Click the **Start application server** link from the first steps panel.
  - Start WebSphere Application Server as a service or from the Start menu, if you are using a Windows operating system.
- View the SystemOut.log file for the application server to verify that the server that supports the administrative console started.
- Check the Web address you use to view the console. By default, this address is http://*server_name*:9060/ibm/console, where *server_name* is the host name.
- If you are browsing the administrative console from a remote machine, try to eliminate connection, address and firewall issues by pinging the server machine from a command prompt, using the server name in the Web address.
- If you have never been able to access the administrative console see Troubleshooting installation.

An Unable to process login. Check user ID and password and try again. error occurs when trying to access the administrative console page

This error indicates that security is enabled for WebSphere Application Server, and that the user ID or password supplied is either not valid or not authorized to access the console.

To access the console:

- If you are the administrator, use the ID defined as the security administrative ID. This ID is stored in the WebSphere Application Server security.xml file.
- If you are not the administrator, ask the administrator to enable your ID for the administrative console.

The directory paths in the administrative console contain strange characters

Directory paths that are used for class paths or resources specified in an assembly tool, in configuration files, or elsewhere that contain strange characters when they are viewed in the administrative console might result from the Java run time interpreting a backslash (\) as a control character.

To resolve this problem, modify Windows-style class paths by replacing occurrences of single back slashes to two. For example, change c:\MyFiles\MyJsp.jsp to c:\\MyFiles\\MyJsp.jsp.

Errors connecting to the administrative console from a browser

This topic describes problems that you can have when logging into the administrative console from a browser.

Review the following information to resolve your browser problem.

If you are able to bring up the browser page, but the console behavior is inconsistent, error-prone, or unresponsive, try upgrading the browser you are using. Older browsers may not support the administrative console's features. For a listing of supported Web browsers, see the Supported hardware and software Web page.

Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Check the following list for your problem and how to solve it:
- When a single user that uses multiple instances of the Mozilla browser logs into the administrative console, the first user ID that logs into the administrative console is the current user.
- A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.
- A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message

When a single user that uses multiple instances of the Mozilla browser logs into the administrative console, the first user ID that logs into the administrative console is the current user.

When a single user logged into an operating system tries to use multiple instances of the Mozilla browser, the first user ID that logs into the administrative console is the current user. This situation occurs because the browser windows share a single process.

To resolve the problem, do one of the following actions:
- Have single users logged into an operating system use a single instance of the Mozilla browser to log into the administrative console.
- If the operating system allows multiple users on an operating system, have each user log into the operating system with a different user ID and bring up a single instance of the Mozilla browser.

A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.

A user on Mozilla browser Version 1.4 selects a check box on a collection table, presses Enter, and receives an error.

To resolve the problem, do one of the following actions:
- Explicitly select a button of interest on the administrative console panel instead of pressing Enter.
- Use a later version of a supported Mozilla browser.
- Use a supported version of the Microsoft Internet Explorer browser.

A user on Mozilla browser Version 1.4 enters an invalid ID or password, presses Enter, and receives an error message

A user on Mozilla browser Version 1.4 enters an invalid user ID or password, presses Enter, and receives an error message. Clicking OK fails to refresh the administrative login screen

To resolve the problem, do one of the following actions:

- Use a later version of a supported Mozilla browser.
- Use a supported version of the Microsoft Internet Explorer browser.

Web server plug-in troubleshooting tips

The following topics might help you diagnose problems with the Web server plug-ins.

If you are having problems using a Web server plug-in, try these steps:

- Review the file *plugins_root*/logs/*web_server_name*/http_plugin.log for clues. Look up any error or warning messages in the message table.
- Review your Web server's error and access logs to see if the Web server is having a problem:
  - IBM HTTP Server and Apache: access.log and error.log.
  - Domino Web server: httpd-log and httpd-error.
  - Sun Java System: access and error.
  - Microsoft IIS: *timedatestamp*.log.

If these files don't reveal the cause of the problem, follow these additional steps.

Plug-in Problem Determination Steps

The plug-in provides very readable tracing which can be beneficial in helping to figure out the problem. By setting the LogLevel attribute in the config/plugin-cfg.xml file to **Trace**, you can follow the request processing to see what is going wrong. At a high level:

1. The plug-in gets a request.
2. The plug-in checks the routes defined in the plugin-cfg.xml file.
3. It finds the server group.
4. It finds the server.
5. It picks the transport protocol, HTTP or HTTPS.
6. It sends the request.
7. It reads the response.
8. It writes it back to the client.

You can see this very clearly by reading through the trace for a single request:

- The first step is to determine if the plug-in has successfully loaded into the Web server.
  - Check to make sure thehttp_plugin.log file has been created.
  - If it has, look in it to see if any error messages indicate some sort of failure that took place during plug-in initialization. If no errors are found look for the following stanza, which indicates that the plug-in started normally. Ensure that the timestamps for the messages correspond to the time you started the Web server:

```
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: ------------System Information----------
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: Bld date: Jul  3 2002, 15:35:09
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: Web server: IIS
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: Hostname = SWEETTJ05
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: OS version 4.0, build 1381, 'Service Pack 6'
[Thu Jul 11 10:59:15 2002] 0000009e 000000b1 - PLUGIN: -----------------------------------------
```

  - Some common errors are:

    lib_security: loadSecurityLibrary: Failed to load gsk library

    Either GSKit did not get installed or the wrong version of GSKit got installed. To determine which situation occurred:

    - Windows On a Windows platform, search for the file gsk7ssl.dll

If you cannot find the appropriate file, try reinstalling the plug-in with the correct GSKit version to see if this fixes the problem.

ws_transport: transportInitializeSecurity: Keyring wasn't set

The HTTPS transport defined in the configuration file was prematurely terminated and did not contain the Property definitions for the keyring and stashfile. Check your XML syntax for the line number given in the error messages that follow this one to make sure the Transport element contains definitions for the keyring and stashfiles before it is terminated.

– If thehttp_plugin.log file is not created, check the Web server error log to see if any plug-in related error messages have been logged there that indicate why the plug-in is failing to load. Typical causes of this can include failing to correctly configure the plug-in with the Web server environment. Check the documentation for configuring the Web server that you are using with the Web server plug-in.

- Determine whether there are network connection problems with the plug-in and the various application servers defined in the configuration. Typically you will see the following message when this is the case:

  ws_common: websphereGetStream: Failed to connect to app server, OS err=%d

  Where %d is an OS specific error code related to why the connect() call failed. This can happen for a variety of reasons.
  – Ping the machines to make sure they are properly connected to the network. If the machines cannot be pinged, there is no way for the plug-in to contact them. Possible reasons for this problem include:
    - Firewall policies that limit the traffic from the plug-in to the application server.
    - The machines are not on the same network.
  – If you are able to ping the machines then the probable cause of the problem is that the port is not active. The port might not be active because the application server or cluster is not started or the application server has gone down for some reason. To verify that this is the problem, you can try to manually telnet into the port that the connect is failing on. If you cannot telnet into the port the application server is not up and that problem needs to be resolved before the plug-in can successfully connect.
- Determine whether other activity on the machines where the servers are installed is impairing the ability of the server to service a request. Check the processor utilization as measured by the task manager, processor ID, or some other outside tool to see if it:
  – Is not what was expected.
  – Is erratic rather than a constant.
  – Shows that a newly added member of the cluster is not being utilized.
  – Shows that a failing member that has been fixed is not being utilized.
- Check the administrative console for server status.

  Check the administrative console to ensure that the application server is started. View the administrative console for error messages or look in the JVM logs.
- In the administrative console, select the problem application server and view its installed applications to verify that they are started.

If none of these steps solves the problem:
- For specific problems that can cause web pages and their contents not to display, seeWeb resource (JSP file, servlet, html file, image, etc) will not display in the information center.
- Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.
- If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

For current information available from IBM Support on known problems and their resolution, see the following topics on the IBM support page:
- HTTP transport
- HTTP plug-in
- HTTP plug-in remote install

For current information available from IBM Support on known problems and their resolution, see the IBM Support page. You should also refer to this page before opening a PMR because it contains documents that can save you time gathering information needed to resolve a problem.

You might find the following topics on the IBM support page helpful:

- HTTP plug-in
- HTTP plug-in remote install

The server process does not start or starts with errors

If a server process does not start or starts with errors, the following topics might help you to diagnose the problem.

Installation program completes successfully, but an application server does not start, or starts with errors

- Browse the Application Server log files for clues. The log files are located by default in:
  - Linux *profile_root*/logs/*server_name*/SystemErr.log and SystemOut.log
  - Windows *profile_root*\logs*server_name*\SystemErr.log and SystemOut.log

  Several applications deployed on an application server or node can take time to start. Browse the SystemOut.log periodically and look at the most recent updates to see if the server is still starting up.
- Linux The `tail -f profile_root/ logs/server_name/SystemOut.log` command is a convenient way to watch the progress of the server.
- Look for any errors or warnings relating to specific resources with the module, such as Web modules, enterprise beans and messaging resources. If you find any, examine the application server configuration file for the configuration settings of that resource. Then restart the server to see if this component causes the problem.

  For example, in a base or non-distributed configuration on Windows systems, browse *profile_root*\config\cells/*ApplicationServerCell*\nodes*node_name*\servers*server_name*\server.xml, and examine the XML tags for the properties of that resource. Change its **initialState** value from START to STOP.
- Look up any error or warning messages in the message reference table by clicking the **Reference** view of the information center navigation and expanding **Messages** in the navigation tree.
- After you create an application server, you must synchronize the nodes before saving the configuration settings for the new server. If you do not synchronize the nodes, your new server might not start.
  1. On the Applications server page listing all of your application server, click **Preferences**.
  2. Select **Synchronize changes with Nodes**, if it is not already selected.
  3. Click **Apply** and then click **Application servers** to return to your list of application servers.
  4. Click **Save** to save the configuration settings for the new server.
- Verify that the logical name that you specified to appear on the console for your application server does not contain invalid characters like: - / \ : * ? " < > and leading or trailing spaces.
- If you are using IBM Cloudscape and receive an `ERROR XSDB6: Another instance of Cloudscape may have already booted the database databaseName` error when starting the application server, consult this topic for more information.
- When using a non-root user ID to run application servers, verify that:
  - The non-root user has write access to the *app_server_root*/temp directory.
  - The JVM has write access to *app_server_root*/config/plugin-cfg.xml file.
  - The non-root user has access to the logs directory.

Message "The socket bind failed for host *hostname* and port *portnumber*. The port may already be in use." occurs when restarting an application server.

The following error message might appear in the SystemOut.log after restarting an application server:

`The socket bind failed for host hostname and port portnumber. The port may already be in use.`

This problem might occur if the network is slow, and the port listed in the message text did not finish listening when the application was stopped and restarted.

To verify that this is the problem, check the port status.

To correct this problem, wait for a few minutes after stopping the server:

1. Verify that no ports are listening. Use the command:
   `netstat -a`
2. Restart the server

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in Troubleshooting help from IBM. Before opening a problem report, see the Support page:

- http://www.ibm.com/software/webservers/appserv/was/support/

The stopServer.sh hangs and creates a Java core dump (Red Hat Linux)

When you run the stopServer.sh script on Red Hat Linux Advanced Server Version 2.1 with the latest operating system patches, it creates a Java core dump and hangs the terminal.

To fix this problem:

- kill all Java and MQ processes
- uninstall the latest version of GNU Standard C++ Library
  - run the command `rpm -e --nodeps libstdc++-2.96-116.7.2`
- Reinstall the older version from Redhat Advanced Server V2.1 CD
  - run the command `rpm -ihv libstdc++-2.96-108.1.rpm`

Problems starting or using the wsadmin command

Use this information if you are having problems starting or using the wsadmin tool.

What kind of problem are you having?

- WASX7016E, WASX7017E, or WASX7209I: Jython scripting language error
- ″WASX7023E: Error creating ″SOAP″ connection to host″ or similar error trying to launch wsadmin command line utility.
- ″com.ibm.bsf.BSFException: error while evaluating Jacl expression: no such method ″<command name>″ in class com.ibm.ws.scripting.AdminConfigClient″ returned from wsadmin command.
- WASX7022E returned from running ″wsadmin -c ...″ command, indicating invalid command.
- com.ibm.ws.scripting.ScriptingException: WASX7025E: String ″″ is malformed; cannot create ObjectName.
- ″The input line is too long″ error returned from the `wsadmin` command on a Windows platform.
- WASX701E: Exception received while running file ″*scriptName*.jacl″; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: missing close-bracket
- WASX7015E: Exception running command: ″source c: ...″; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: couldn’t read file ″c: ...″
- Unexpected error CWSIV0806E in WebSphere log following deletion of an outbound service
- Separator exception
- Enabling authentication in the file transfer service
- The format of ″$AdminConfig list″ output has changed for V6.0
- You are not prompted for user ID and password after applying V6.0.2 service if you use an existing 6.0 profile
- Required ports not defined when a new V5.0/V5.1 server is created

- When running the $AdminApp searchJNDIReferences command with the Java Naming and Directory Interface (JNDI) name of a message destination, the message destination reference is not returned
- AWXJR0006E: Exception received when configuring JACC provider for the Tivoli Access Manager

If you do not see your problem here:
- If you are not able to enter wsadmin command mode, try running **wsadmin -c** ″**$Help wsadmin**″ for help in verifying that you are entering the command correctly.
- If you can get the wsadmin command prompt, enter **$Help help** to verify that you are using specific commands correctly.
- wsadmin commands are a superset of Jacl (Java Command Language), which is in turn a Java-based implementation of the Tcl command language. For details on Jacl syntax beyond wsadmin commands, refere to the Tcl developers' site, http://www.tcl.tk. For specific details relating to the Java implementation of Tcl, refer to http://www.tcl.tk/software/java.
- Browse the *install_dir*/profiles/*profile_name*/logs/wsadmin.traceout file for clues.
  - Keep in mind that wsadmin.traceout is refreshed (existing log records are deleted) whenever a new wsadmin session is started.
  - If the error returned by wsadmin does not seem to apply to the command you entered, for example, you receive WASX7023E, stating that a connection could not be created to host ″myhost,″ but you did not specify ″-host myhost″ on the command line, examine the properties files used by wsadmin to determine what properties are specified. If you do not know what properties files were loaded, look for the WASX7326I messages in the wsadmin.traceout file; there will be one of these messages for each properties file loaded.

If none of these steps fixes your problem, 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 please contact IBM support.

WASX7016E, WASX7017E, or WASX7209I: Jython scripting language error

The following errors may occur when you run this Jython script:

Jython script

```
″profile_root/bin/wsadmin.sh -lang jython -profile profile_name -host host_name -f
script_file.py″
```

Error Messages

```
WASX7209I: Connected to process "server1" on node
node_name using SOAP connector;  The type of process is:
UnManagedProcess
```

```
WASX7016E: Exception received while reading file
"script_file.py"; exception information:
sun.io.MalformedInputException
```

```
WASX7017E: Exception received while running file
"script_file.py"; exception information:
com.ibm.bsf.BSFException: exception from Jython: Traceback
(innermost last):   File "<string>" line 89, in ? NameError: log
```

These errors can occur because there are UTF-8 characters in the file that are not valid. The default codepage for RHEL 3 is UTF-8 (en_US.UTF-8). When doing a text file read through Java™ code, the program assumes all characters are UTF-8 encoded. There might be one or more characters in the file that are not part of the UTF-8 specification, causing the load to fail. An easy way to determine if a character that is not valid is causing the error is to enter export LANG=C and run the script again. If you determine that the problem is a character that is not valid:

1. Open a new text reader on the file.
2. Read it one character at a time.

3. Print the character that is not valid.
4. When you press the back characters, you get the exception and will then know which of the characters is causing the error.
5. Remove any characters that are not valid, then run the script again

″WASX7023E: Error creating ″SOAP″ connection to host″ or similar error trying to launch wsadmin command line utility

By default, the wsadmin utility attempts to connect to an application server at startup. This is because some commands act upon running application servers. This error indicates that no connection could be established.

To resolve this problem:

- If you are not sure whether an application server is running, start it by entering **startserver *servername*** from the command prompt. If the server is already running, you will see an error similar to ″ADMU3027E: An instance of the server is already running″.
- If you are running a Network Deployment configuration, you will first need to start the deployment manager by running ″startManager″ or ″startManager.sh″ from the *install_dir*/bin directory. Then you can launch wsadmin immediately to connect to the deployment manager, or start a node and application server to connect to.
- If an application server is running and you still get this error:
  - If you are running remotely (that is, on a different machine from the one running WebSphere Application Server), you must use the **-host *hostname*** option to the wsadmin command to direct wsadmin to the right physical server.
  - If you are using the -host option, try pinging the server machine from the command line from the machine on which you are trying to launch wsadmin to verify there are no issues of connectivity such as firewalls.
  - verify that you are using the right port number to connect to the WebSphere Application Server process:
    - If you are not specifying a port number (using the -port option) when you start the wsadmin tool, the wsadmin tool uses the default port specified in *install_dir*/profiles/*profile_name*/properties/wsadmin.properties file, property name=com.ibm.ws.scripting.port (default value =8879).
    - The port that wsadmin should send on depends on the server process wsadmin is trying to connect to.

      For a single-server installation, wsadmin attempts to connect to the application server process by default. To verify the port number:
      - Look in the file *profile_root*/config/cells/*node_name*/nodes/*node_name*/serverindex.html for a tag containing the property **serverType=″APPLICATION_SERVER″**.
      - Look for an entry within that tag with the property **endPointName=″SOAP_CONNECTOR_ADDRESS″**.
      - Look for a **port** property within that tag. This is the port wsadmin should send on.

      In a Network Deployment installation, wsadmin launched from the bin directory on the Network Deployment installation attempts to send requests to the deployment manager by default. To verify the port number:
      - Get the hostname of the node on which the Deployment Manager is installed.
      - Using that hostname, look in *profile_root*/config/cells/*cell_name*/nodes/*node_name*/serverindex.html file for a tag containing the property **serverType=″DEPLOYMENT_MANAGER″**.
      - Within that tag, look for an entry with a property **endPointName=″SOAP_CONNECTOR_ADDRESS″**.
      - Within that tag, look for a ″port″ property. This is the port that the wsadmin tool should send on.

″com.ibm.bsf.BSFException: error while eval'ing Jacl expression: no such method *command name* in class com.ibm.ws.scripting.AdminConfigClient″ returned from wsadmin command.

This error is usually caused by a misspelled command name. Use the **$AdminConfig help** command to get information about what commands are available. Note that command names are case-sensitive.

WASX7022E returned from running ″wsadmin -c ...″ command, indicating invalid command

If the command following -c appears to be valid, the problem may be caused by the fact that on Unix, using wsadmin -c to invoke a command that includes dollar signs results in the shell attempting to do variable substitution. To confirm that this is the problem, check the command to see if it contains an unescaped dollar sign, for example: **wsadmin -c** ″**$AdminApp install ....**″.

To correct this problem, escape the dollar sign with a backslash. For example: **wsadmin -c** ″**\$AdminApp install ...**″.

com.ibm.ws.scripting.ScriptingException: WASX7025E: String ″″ is malformed; cannot create ObjectName

One possible cause of this error is that an empty string was specified for an object name. This can happen if you use one scripting statement to create an object name and the next statement to use that name, perhaps in an ″invoke″ or ″getAttribute″ command, but you don't check to see if the first statement really returned an object name. For example (the following samples use basic Jacl commands in addition to the wsadmin Jacl extensions to make a sample script):

```
#let's misspell "Server"
set serverName [$AdminControl queryNames type=Srever,*]
$AdminControl getAttributes $serverName
```

To correct this error, make sure that object name strings have values before using them. For example:

```
set serverName[$AdminControl queryNames node=mynode,type=Server,name=server1,*]
if {$serverName == ""} {puts "queryNames returned empty - check query argument"}
else {$AdminControl getAttributes $serverName}
```

For details on Jacl syntax beyond wsadmin commands, refer to the Tcl developers' site, http://www.tcl.tk.

″The input line is too long″ error returned from the wsadmin command on a Windows platform

This error indicates that the Windows command line limit of 2048 characters has been exceeded, probably due to a long profile path used within the **wsadmin.bat** command. You may get this error when running wsadmin in a Windows command prompt or calling wsadmin from a `.bat` file, an `ant` build file, or Profile Management Tool. If this error results in running wsadmin other than from the Profile Management Tool, avoid the problem by using the Windows **subst** command, which allows you to map an entire path to a virtual drive. To see the syntax of the **subst** command, enter `help subst` from a Windows command prompt.

For example, if the product resides in the *app_server_root* directory, edit the *app_server_root*\bin\setupCmdLine.bat file as follows:

```
SET CUR_DIR=%cd%
cd /d "%~dp0.."
SET WAS_HOME=%cd%
cd /d "%CUR_DIR%"

@REM add the following two lines to workaround Windows 2K command line length limit
subst w: %WAS_HOME%
```

```
set WAS_HOME=w:

...
...
```

Then edit the `setupCmdLine.bat` file residing in the `bin` directory of your profile as follows:

```
SET WAS_USER_PROFILE=...
SET USER_INSTALL_ROOT=...
SET WAS_HOME=app_server_root
SET JAVA_HOME=app_server_root\java

@REM add the following three lines to workaround Windows 2K command line length limit
subst w: %WAS_HOME%
set WAS_HOME=w:
set JAVA_HOME=%WAS_HOME%\java

...
...
```

If this error occurred while running the Profile Management Tool, you have to rerun the Profile Management Tool to provide a shorter profile path with a shorter profile name. If this does not fix the problem, follow the same instructions above to edit the setupCmdLine.bat file in the bin directory of your WebSphere Application Server installation. After editing the file, rerun the Profile Management Tool. If the same problem persists, reinstall WebSphere Application Server with a shorter installation root directory path.

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.

WASX701E: Exception received while running file *"scriptName*.jacl*"*; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: missing close-bracket

This error is caused by a mix-up between the code page that the scripting client expects to see and the code page in which the Jacl script was written.

To fix this problem, set the `-Dscript.encoding=script codepage` option in the wsadmin.sh or wsadmin.bat file to the code page of the Jacl script. The following guideline will help you to determine the code page of the script:

- If the script was written in the OMVS interface using the OEDIT editor, the code page is IBM-037. In this case, set the option to the following: `-Dscript.encoding=Cp037`
- If the script was written in a telnet session to the OMVS interface using the VI editor, the code page is IBM-1047. In this case, set the option to the following: `-Dscript.encoding=Cp1047`
- IF the script was written on a personal computer, or any other ASCII machine, and was transferred to the host as a text file, the code page is IBM-1047. In this case, set the option to the following: `-Dscript.encoding=Cp1047`
- If the script was written on a personal computer, or any other ASCII machine, and transferred to the host in binary format, the code page is ISO-8859-1 (ASCII). In this case, you do not need to set the option because the default is ASCII. You should review other possible reasons for this error.

WASX7015E: Exception running command: *"*source c: ...*"*; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: couldn't read file *"*c: ...*"*

This error is caused by using a backslash (\ ) instead of a forward slash (/) when running the wsadmin command to source a Jacl script in a Windows® environment. The file path cannot contain the backslash (

\ ); for example, **wsadmin> source c:\temp\test.jacl**. The file path must use the forward slash (/) as the path separator; for example, **wsadmin> source c:/temp/test.jacl**.

To correct this problem use the forward slash (/) in the file path when using the wsadmin command to source a Jacl script in a Windows® environment:

```
app_server_root\bin>wsadmin
WASX7209I: Connected to process "dmgr" on node sunCellManager01
using SOAP connector;  The type of process is:
DeploymentManager WASX7029I: For help, enter: "$Help help"
wsadmin>source c:/temp/test.jacl
```

Unexpected error CWSIV0806E in WebSphere log following deletion of an outbound service

This error occurs when an exception is issued for destination MPOutBoundServicePortDestination, on messaging engine trueliesNode01.server1-FVTSIBus01, on bus FVTSIBus01, for endpoint activation:

```
com.ibm.websphere.sib.exception.SINotPossibleInCurrentConfigurationException: CWSIP0111E: The
destination with name MPOutBoundServicePortDestination is being deleted on messaging engine {1}.
```

You can ignore this error; it is benign.

Separator exception

You must use forward slashes (/) as your path separator. Backward slashes (\) will not work.

Enabling authentication in the file transfer service

In WebSphere Application Server Network Deployment Version 5.0.1 or later, the file transfer service was enhanced to provide role-based authentication. Two versions of the file transfer Web application were provided. By default, the version that does not authenticate its caller is installed. This default supports compatibility between the WebSphere Application Server Network Deployment, Version 5.0 and Version 5.0.1 or later. Turning the file transfer authentication on is recommended to prevent unauthorized use of the file transfer application. However, if you have any Version 5.0 clients in your WebSphere Application Server Network Deployment environment, they will not be able to communicate with the secured file transfer application if global security is turned on.

In WebSphere Application Server Version 6.x, mixed cells are supported and file transfer has become a system application. If all of the nodes in the cell are of Version 5.0.1 or later, you can activate authentication in the file transfer service by redeploying the file transfer application at the deployment manager. The compatible version is shipped in the ${*app_server_root*}/systemApps/filetransfer.ear directory. The secured version is provided in the ${*app_server_root*}/systemApps/filetransferSecured.ear directory.

Also in WebSphere Application Server Version 6.x, the file transfer service is also supported for WebSphere Application Server and WebSphere Application Server Express as long as the node is not federated into a managed cell. If this node becomes federated, it will make use of the deployment manager file transfer.

A wsadmin Jacl script is provided to help you redeploy file transfer. The script is redeployFileTransfer.jacl and you can find it in the ${*app_server_root*}/bin directory. After the deployment manager and all the nodes are at Version 5.0.1 or later, you can deploy the secured file transfer service by running the script. The syntax for running the script from the bin directory includes the following:

```
wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationXxx
cell_name node_name server_name
```

where ″Xxx″ is ″On″ or ″Off″.

For example, when running the script to enable use of the filetransferSecured.ear, the syntax is similar to the following example:

```
wsadmin -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOn managedCell managedCellManager dmgr"
```

or

```
wsadmin -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOn baseCell base server1"
```

If you want to go back to run the file transfer service without authentication, you can run the script as shown in the following example:

```
wsadmin -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOff managedCell managedCellManager dmgr"
```

or

```
wsadmin -profile redeployFileTransfer.jacl -c
"fileTransferAuthenticationOff baseNodeCell baseNode server1"
```

The format of ″$AdminConfig list″ output has changed for V6.0

If you have a script that parses the output of $AdminConfig list, such as $AdminConfig list Node, you might receive errors, such as ″Node not found.″ Scripts should not parse the output of $AdminConfig; however, if you have a script that does this parsing, it must be updated for WebSphere Application Server V6.0 to reflect changes to the output format.

You are not prompted for user ID and password after applying V6.0.2 service if you use an existing 6.0 profile

If security is enabled, executing a .bat file requires a user ID and password. On V6.0.2, a new feature is introduced to prompt you for a user ID and password if they are not supplied in the command line. However, this feature is not available for profiles that were created at the 6.0 level.

Property files for profiles created at the V6.0 level are not updated after applying the V6.0.2 refresh pack.

There are two solutions to this problem:

1. Create a new profile after applying the V6.0.2 service. This new profile contains all the updated property files and you will then be prompted for a user ID and password.
2. If you want to keep the existing V6.0 profile and use the new prompt feature, you must manually update three files:
   - for *app_server_root*/properties/soap.client.props, add the following line:

     ```
     com.ibm.SOAP.loginSource=prompt
     ```
   - for *app_server_root*/properties/wsjaas_client.conf, add the following lines:

     ```
     WSAdminClientLogin {
       com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy required del
     egate=com.ibm.ws.security.common.auth.module.WSAdminClientLoginModuleImpl;
     };
     ```
   - for *app_server_root*/bin/setupCmdLine.bat add the following line:

     ```
     SET JAASSOAP=-Djava.security.auth.login.config=app_server_root/properties/
     wsjaas_client.conf
     ```

Required ports not defined when a new V5.0/V5.1 server is created

Required ports not defined when a new V5.0/V5.1 server is created

You create a new V5.0 or V5.1 server using a default template in a mixed cell environment, and all of the required ports are not defined for the newly created server.

Add the following two entries to the serverindex.xml directory in the deployment manager profile: *profile_root*/config/templates/servertypes/APPLICATION_SERVER/serverindex.xml directory:

- ```
  <serverEntries xmi:id="ServerEntry_3" serverName="default_5X"
  serverType="APPLICATION_SERVER">
  <specialEndpoints xmi:id="NamedEndPoint_25" endPointName="BOOTSTRAP_ADDRESS">
  <endPoint xmi:id="EndPoint_25" host="$(node.host.name)" port="2809"/>
  </specialEndpoints>
  <specialEndpoints xmi:id="NamedEndPoint_26" endPointName="SOAP_CONNECTOR_ADDRESS">
  <endPoint xmi:id="EndPoint_26" host="$(node.host.name)" port="8880"/>
  </specialEndpoints>
  <specialEndpoints xmi:id="NamedEndPoint_27" endPointName="SAS_SSL_
  SERVERAUTH_LISTENER_ADDRESS">
  <endPoint xmi:id="EndPoint_27" host="$(node.host.name)" port="0"/>
  </specialEndpoints>
  <specialEndpoints xmi:id="NamedEndPoint_28" endPointName="CSIV2_
  SSL_SERVERAUTH_LISTENER_ADDRESS">
  <endPoint xmi:id="EndPoint_28" host="$(node.host.name)" port="0"/>
  </specialEndpoints>
  <specialEndpoints xmi:id="NamedEndPoint_29" endPointName="CSIV2_
  SSL_MUTUALAUTH_LISTENER_ADDRESS">
  <endPoint xmi:id="EndPoint_29" host="$(node.host.name)" port="0"/>
  </specialEndpoints>
  </serverEntries>
  ```
- ```
  <serverEntries xmi:id="ServerEntry_4" serverDisplayName="default
  ZOS_5X" serverName="defaultZOS_5X" serverType="APPLICATION_SERVER"
  serverUniqueId="BB80B67909190083000000DC0010200209390F08">
  <specialEndpoints xmi:id="NamedEndPoint_31" endPointName="BOOTSTRAP_ADDRESS">
  <endPoint xmi:id="EndPoint_31" host="$(node.host.name)" port="2809"/>
  </specialEndpoints>
  <specialEndpoints xmi:id="NamedEndPoint_32" endPointName="SOAP_CONNECTOR_ADDRESS">
  <endPoint xmi:id="EndPoint_32" host="$(node.host.name)" port="8880"/>
  </specialEndpoints>
  <specialEndpoints xmi:id="NamedEndPoint_33" endPointName="ORB_SSL_LISTENER_ADDRESS">
  <endPoint xmi:id="EndPoint_33" host="*" port="0"/>
  </specialEndpoints>
  <specialEndpoints xmi:id="NamedEndPoint_34" endPointName="ORB_LISTENER_ADDRESS">
  <endPoint xmi:id="EndPoint_34" host="*" port="0"/>
  </specialEndpoints>
  </serverEntries>
  ```

The xmi:ids must all be unique within this file.

When running the $AdminApp searchJNDIReferences command with the Java Naming and Directory Interface (JNDI) name of a message destination, the message destination reference is not returned

This problem occurs when the command $AdmnApp searchJNDIReferences is run with the JNDI name of a message destination. The command cannot collect the message destination reference that is defined in the application deployment descriptor. The message destination that you configured for the application server is defined with a message destination link on not one element, but two: both a message-driven bean (MDB) and a message destination reference.

Currently there is no workaround for this problem. The $AdmnApp searchJNDIReferences command cannot return a reference for a message destination that is defined on two elements.

AWXJR0006E: Exception received when configuring JACC provider for the Tivoli Access Manager

This problem occurs when you attempt to configure a JACC provider for the Tivoli Access Manager using the wsdamin tool in a deployment manager environment with or without nodes added. You enter a deployment manager node name, for example, `t54Manager`, instead of an asterisk (*) for all nodes. The wsadmin command finishes successfully but when you try to add a new node and start the node, or start an existing node, you receive an error in the nodeagent SystemOut.log file similar to the following:

```
[12/7/05 17:09:51:266 CST] 0000000a SystemOut     O AWXJR0006E   The file,
C:\cc_was602\WebSphere\AppServer\profiles\AppSrv01\etc\tam\amwas.t54Node01_.amjacc.pr
operties, was not found.
[12/7/05 17:09:51:266 CST] 0000000a distSecurityC E   SECJ0391E: Error when setting
the Policy object to the providers policy implementation {0}. The exception is
{1}.
[12/7/05 17:09:51:281 CST] 0000000a distSecurityC E   SECJ0324E: Error during Java
2 Security and Dynamic Policy initialization.
```

To workaround this problem, unconfigure the existing Tivoli Access Manager configuration and configure it again using an asterisk (*) for the node name, for example:

```
wsadmin.bat -user wsadmin -password pw1 -f enableTAM.jacl "*" TAMHostName:7135"
TAMHostName:7136:1"   "cn=wsadmin,o=ibm,c=us" "o=ibm,c=us" "sec_master"
sec_master pw1 "9990:9999"
```

Problems using command line tools

This article provides troubleshooting support for a variety of problems relating to using command line tools.

What kind of problem are you having?

- Just-in-time (JIT) compiler is disabled when you start application server with DEBUG enabled on a Red Hat Linux machine
- The startServer.sh or stopServer.sh commands fail to start or stop the server when the server definition is part of the configuration repository.
- With Windows service, there is no indication when a server is already started.
- ″ADMU0125E: Change the clock of the new node to be within {0} minutes of the clock of the deployment manager″ error message occurs during federation

Just-in-time (JIT) compiler is disabled when you start the application server with DEBUG enabled on a Red Hat Linux machine

The just-in-time (JIT) compiler is disabled when you start the application server with Software Developer Kit (SDK) DEBUG enabled on a Red Hat Linux® machine, even though JIT is set to enabled. To verify this setting, check the SystemOut.log or the startServer.log file.

Use the administrative console to remove the following DEBUG options of the Java™ process definition. -Xdebug -Xnoagent

For more information, see the V6.0 Information Center Release Notes page.

The startServer.sh or stopServer.sh commands fail to start or stop the server when the server definition is part of the configuration repository.

This problem occurs when the startServer.sh or stopServer.sh commands are trying to start or stop non Java™ process. To solve this problem, use the -nowait option to start or stop the server, for example:

```
startServer.sh webserver1 -nowait
```

```
stopServer.sh webserver1 -nowait
```

With Windows service, there is no indication when a server is already started.

When attempting to start an already-started server from the command line, there is no indication that the server is already started and running. When running startManager.bat on Windows® the following output is displayed before the command returns:

```
ADMU7701I: Because dmgr is registered to run as a Windows Service, the request to start this
server will be completed by starting the associated Windows Service.
```

When running startServer.bat, the following output is displayed before the command returns:

```
ADMU7701I: Because server1 is registered to run as a Windows Service, the request to start this
server will be completed by starting the associated Windows Service.
```

When running WASService.exe, the following output is displayed before the command returns:

```
Starting Service: service name
```

To check if the server is started or if the service is running, use the `serverStatus` *`server name`* command or the `WASService -status` *`service name`* command.

″ADMU0125E: Change the clock of the new node to be within {0} minutes of the clock of the deployment manager″ error message occurs during federation

The workaround for this problem is to adjust the time on the node to be within the recommended amount of the deployment manager. Verify that the time zones are correct and that the times within the time zones are correct. For AIX systems, if the time on the node system is within five minutes of the deployment manager, review the `timezone` setting in the /etc/environment file. Verify that the TZ= property is set correctly. For example, in the Central Time Zone, it should read TZ=CST6CDT. For more information, see the environment file section of the Files Reference Web site.

If none of these steps fixes your problem, 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.

Problems using tracing, logging or other troubleshooting features

Use this information if you are having problems using tracing, logging or other troubleshooting tools.

What kind of problem are you having?
- Netscape browser fails when trying to enable a component trace.

Netscape browser fails when trying to enable a component trace

On systems using AIX, the Netscape browser fails when you try to enable trace on a component.

To work around this problem, do one of the following:
- Disable JavaScript on the browser and continue setting trace.
- Administer the AIX server from a remote machine running another browser and operating system.
- Change the trace manually in the *server.xml* file.

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server - Express.

Default product locations when the root user or an administrator user installs the product

The root user or administrator user (on a Windows system) is capable of registering shared products and installing into system-owned directories. The following default directories are system-owned directories.

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server - Express products or components, of course, require multiple locations.

app_server_root - the install_root for WebSphere Application Server

The following list shows default installation root directories for WebSphere Application Server - Express:

AIX /usr/IBM/WebSphere/AppServer

HP-UX /opt/IBM/WebSphere/AppServer

Linux /opt/IBM/WebSphere/AppServer

Solaris /opt/IBM/WebSphere/AppServer

Windows C:\Program Files\IBM\WebSphere\AppServer

profile_root

The following list shows the default directory for a profile named *profile_name* on each distributed operating system:

AIX /usr/IBM/WebSphere/AppServer/profiles/*profile_name*

HP-UX /opt/IBM/WebSphere/AppServer/profiles/*profile_name*

Linux /opt/IBM/WebSphere/AppServer/profiles/*profile_name*

Solaris /opt/IBM/WebSphere/AppServer/profiles/*profile_name*

Windows C:\Program Files\IBM\WebSphere\AppServer\profiles*profile_name*

plugins_root

The following default installation root is for the Web server plug-ins for WebSphere Application Server:

AIX /usr/IBM/HTTPServer/Plugins

HP-UX /opt/IBM/HTTPServer/Plugins

Linux /opt/ibm/HTTPServer/Plugins

Solaris /opt/IBM/HTTPServer/Plugins

Windows C:\Program Files\IBM\HTTPServer\Plugins

web_server_root

The following default installation root directories are for the IBM HTTP Server:

AIX /usr/IBM/HTTPServer

HP-UX /opt/IBM/HTTPServer

Linux /opt/ibm/HTTPServer

Solaris /opt/IBM/HTTPServer

Windows C:\Program Files\IBM\HTTPServer

gskit_root

The following list shows the default installation root directories for Version 7 of the IBM Global Security Kit (GSKit):

AIX /usr/ibm/gsk7

HP-UX /opt/ibm/gsk7

Linux /opt/ibm/gsk7

Solaris /opt/ibm/gsk7

Windows C:\Program Files\IBM\GSK7

app_client_root

The following default installation root directories are for the WebSphere Application Client:

AIX /usr/IBM/WebSphere/AppClient (J2EE Application client only)

HP-UX /opt/IBM/WebSphere/AppClient (J2EE Application client only)

Linux /opt/IBM/WebSphere/AppClient (J2EE Application client only)

Solaris /opt/IBM/WebSphere/AppClient (J2EE Application client only)

Windows C:\Program Files\IBM\WebSphere\AppClient

updi_root

The following list shows the default installation root directories for the Update Installer for WebSphere Software:

AIX /usr/IBM/WebSphere/UpdateInstaller

HP-UX /opt/IBM/WebSphere/UpdateInstaller

Linux /opt/IBM/WebSphere/UpdateInstaller

Solaris /opt/IBM/WebSphere/UpdateInstaller

Windows C:\Program Files\IBM\WebSphere\UpdateInstaller

cip_app_server_root

The following list shows the default installation root directories for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server - Express product bundled with one or more maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts:

AIX /usr/IBM/WebSphere/AppServer/cip/*cip_uid*

HP-UX /opt/IBM/WebSphere/AppServer/cip/*cip_uid*

Linux /opt/IBM/WebSphere/AppServer/cip/*cip_uid*

Solaris /opt/IBM/WebSphere/AppServer/cip/*cip_uid*

Windows C:\Program Files\IBM\WebSphere\AppServer\cip*cip_uid*

The *cip_uid* variable is the CIP unique ID generated during creation of the build definition file. You can override the generated value in the Build definition wizard. Use a unique value to allow multiple CIPs to install on the system.

component_root

The component installation root directory is any installation root directory described in this topic. Some programs are for use across multiple components. In particular, the Update Installer for WebSphere Software is for use with WebSphere Application Server - Express, Web server plug-ins, the Application Client, and the IBM HTTP Server. All of these components are part of the product package.

Default product locations when a non-root user or a non-administrator user installs the product

The non-root user or non-administrator user (on a Windows system) is not capable of registering shared products and installing into system-owned directories. The following default directories are user-owned directories in the home directory of the non-root installer as opposed to being globally shared resources that are available to all users.

app_server_root

The following list shows the default installation directories for non-root installation of WebSphere Application Server - Express:

AIX *user_home*/IBM/WebSphere/AppServer

HP-UX *user_home*/IBM/WebSphere/AppServer

Linux *user_home*/IBM/WebSphere/AppServer

Solaris *user_home*/IBM/WebSphere/AppServer

Windows C:\IBM\WebSphere\AppServer

profile_root

The following list shows the default directories for creating profiles:

AIX *user_home*/IBM/WebSphere/AppServer/profiles/

HP-UX *user_home*/IBM/WebSphere/AppServer/profiles/

Linux *user_home*/IBM/WebSphere/AppServer/profiles/

Solaris *user_home*/IBM/WebSphere/AppServer/profiles/

Windows C:\IBM\WebSphere\AppServer\profiles\

web_server_root

The following default installation root directories are for the IBM HTTP Server:

AIX *user_home*/IBM/HTTPServer

HP-UX *user_home*/IBM/HTTPServer

Linux *user_home*/ibm/HTTPServer

Solaris *user_home*/IBM/HTTPServer

Windows C:\IBM\HTTPServer

plugins_root

The following list shows the default installation root directories for the Web server plug-ins for WebSphere Application Server:

user_home/IBM/HTTPServer/Plugins

user_home/IBM/HTTPServer/Plugins

user_home/ibm/HTTPServer/Plugins

user_home/IBM/HTTPServer/Plugins

C:\IBM\HTTPServer\Plugins

app_client_root

The following list shows the default installation root directories for the WebSphere Application Client:

user_home/IBM/WebSphere/AppServer/AppClient (J2EE Application client only)

user_home/IBM/WebSphere/AppClient (J2EE Application client only)

user_home/IBM/WebSphere/AppClient (J2EE Application client only)

user_home/IBM/WebSphere/AppClient (J2EE Application client only)

C:\IBM\WebSphere\AppClient

updi_root

The following list shows the default installation directories for non-root installation of WebSphere Application Server - Express:

user_home/IBM/WebSphere/UpdateInstaller

user_home/IBM/WebSphere/UpdateInstaller

user_home/IBM/WebSphere/UpdateInstaller

user_home/IBM/WebSphere/UpdateInstaller

C:\Program Files\IBM\WebSphere\UpdateInstaller

cip_app_server_root

The following list shows the default installation root directories for a WebSphere Application Server - Express product CIP:

user_home/IBM/WebSphere/AppServer/cip/*cip_uid*

user_home/IBM/WebSphere/AppServer/cip/*cip_uid*

user_home/IBM/WebSphere/AppServer/cip/*cip_uid*

user_home/IBM/WebSphere/AppServer/cip/*cip_uid*

C:\IBM\WebSphere\AppServer\cip*cip_uid*

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

For trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).