WebSphere® Application Server for z/OS, Version 6.0.1

Administering applications and their environment

GA22-7962-03

Note

Before using this information, be sure to read the general information under “Notices” on page 1699.

Compilation date: March 16, 2005

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

What is new for administrators

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

"Introduction: System administration" on page 3

This topic describes the administration of WebSphere Application Server, Version 6 products and the applications that run on them.

Presentations from IBM Education Assistant

The following presentations provide a quick overview:

- System management architecture
- Administrative security
- Administrative clients overview
  - Start, stop, and monitor processes
  - Other commands
  - Browser-based administrative console
  - Scripting - wsadmin
- Topologies and logical administrative domains
  - Resource scoping
  - Cells, deployment managers, and node agents
  - Build cells - Add and remove nodes
  - Manage node groups
- Applications and application resources
  - Application management overview
  - JDBC
  - Installing and uninstalling applications
  - Managed application resources - Enhanced EAR files
  - Fine grained application updates
- Servers
  - Manage Web server nodes
- Configuration management
  - Configuration repository
  - Configuration archives
  - File synchronization

Contents of this section: Administering applications and their environments

Setting up the application serving environment

This section is for the administrator who is responsible for integrating application serving capabilities into an existing network environment. It looks at the product as part of a larger system, typically a production environment or realistic test environment. This section reiterates some installation and customization activities, including topology planning and creating product configurations. It carries the focus into the administrative realm, discussing port configuration and other network concerns. See also ″Overview and new features for installing an application serving environment.

This information expands the topology planning discussion by describing how to set up and maintain logical administrative domains of cells and nodes, and how to balance workload through clustering and high availability configurations.

Using the administrative clients
This section describes the many options available for administering your applications and the servers to which the applications are deployed. Options include the graphical administrative console; scripting with the wsadmin tool; programmatic administration using Java Management Extensions (JMX) and MBeans; and a wide array of command-line tools, including ANT.

Chapter 3, "Starting and stopping quick reference," on page 23
This section summarizes what can be started and stopped, including applications and the application servers on which these applications are deployed.

Chapter 4, "Class loading," on page 25
This section describes how to configure class loaders. It includes both configuration that is performed during application assembly (packaging) and configuration performed at the server. The product run-time environment uses class loaders to find and load new classes for an application. Class loaders are part of the Java virtual machine (JVM) code and are responsible for finding and loading class files.

Chapter 5, "Deploying and administering applications," on page 37
This section describes how to deploy applications onto application servers, and then how to administer the deployed applications. It includes installing applications, starting applications, exporting application files, updating applications, removing applications, and other common tasks.

Administer WebSphere applications
This section provides administrative instructions that are specific to the various types of applications. For example, you can focus on administering your Web applications in their Web container; or aspects of Web services support; or the messaging or security subsystems.

Chapter 7, "Troubleshooting deployment," on page 1539
This section describes how to identify and handle a variety of problems encountered during development, assembly, and deployment activities.

Chapter 8, "Troubleshooting administration," on page 1551
This section describes how to identify and handle a variety of problems encountered during administrative activities.

Getting started with WebSphere Application Server

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

Installing

See ″Task overview: installing″ for a description of installing the WebSphere Application Server product and other installable components.

Configuring

See ″Configuring the product after installation″ in the information center for a description of what to do after installing the product.

Migrating

See ″Migrating and coexisting″ for a description of how to migrate applications and configuration data from a previous version of WebSphere Application Server.

Using the Samples Gallery

See ″Accessing the Samples (Samples Gallery)″ in the information center for a description of the set of Samples that ship with each product. The Samples demonstrate common Web application tasks.

Deploying applications

The information center describes how to deploy Web components, such as servlets and JSP files.

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” on page 4.

- **Administrative agents**

  Servers, nodes and node agents, cells and the deployment manager are fundamental concepts in the administrative universe of the product. It is also important to understand the various processes in the administrative topology and the operating environment in which they apply.

  For more information, refer to “Welcome to basic administrative architecture” on page 9.

- **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 7.

- **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 8.

- **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 8.

- **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 9.

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.

The information center topic, ″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.

Use both the MVS console and the ″Application Server administrative console″ to administer the Application Server. For example:

- Use MVS commands that are issued from the MVS console to start the base application server controller region, and the node agent and deployment manager.
- In an application server configuration, you must start the first server with an MVS operator command. After the first server is started, you can use the administrative console, if it has this application, to start other application servers in the node. After the deployment manager and node agent are active (in an ND configuration), you can use the administrative console to start and stop application servers.
- Workload management starts all servant regions using Address Space Create (ASCRE) with the administrative console, you can display and modify Application Server applications and the environments in which they run.

Identifying where to perform WebSphere Application Server operations

Administering WebSphere Application Server involves the use of both the MVS console and the WebSphere Application Server administrative console. For example:

- Use MVS commands issued from the MVS console to start the base Application Server control region, the network deployment node agent, and the deployment manager.
- In a base Application Server configuration, you must start the first server with an MVS operator command. Once the first server starts, you can then use the administrative console, if it has this application, to start other Application Servers in the node. Once the deployment manager and node agent are active (in a network deployment configuration), you can use the administrative console to start and stop application servers.
- Workload management starts all servant regions using Address Space Create (ASCRE).

The following table lists the main Application Server operations tasks and directs you to information that helps you to perform these tasks. The Application Server activities and operations can be performed from:

- A z/OS or OS/390 MVS console (most operations)
- The Application Server administrative console (some operations)
- TSO or resource recovery services (RRS) panels (some operations).

Table 1. Application Server operations tasks

Task	MVS console	Application Server administrative console	TSO panel	Reference to associated procedure
Start operations				
Starting the Application Server environment and location service daemon	Yes	No	No	See ″Starting servers″ in the information center.
Starting a cluster or application server	Yes	Application server only	No	See ″Starting clusters″, ″Starting servers″, and Using the administrative console in the information center.
Stop operations				
Stopping the location service daemon	Yes	No	No	See Steps for stopping or canceling the location service daemon from the MVS console .
Stopping a cluster	Yes	Yes	No	See ″Stopping clusters″ in the information center.
Stopping an application server	Yes	Yes	No	See ″Stopping servers″ in the information center.
Cancel operations				
Canceling the location service daemon	Yes	No	No	See Steps for stopping or canceling the location service daemon from the MVS console.
Canceling a cluster	Yes	Yes	No	See ″Stopping clusters″ in the information center.
Canceling an application server	Yes	Yes	No	See ″Stopping servers″ in the information center.
Display operations				
Displaying the status of ARM-registered address spaces including clusters and servants	Yes	No	No	See ″Displaying the status of ARM-registered address spaces including WebSphere Application Server for z/OS and server instances″ in the information center.
Displaying units of work (threads) for DB2	Yes	No	No	See *DB2 Universal Database for OS/390 and z/OS Command Reference* athttp://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi.
Displaying indoubt units of work (threads) for DB2	Yes	No	No	See *DB2 Universal Database for OS/390 and z/OS Command Reference* athttp://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi.
Displaying units of work for RRS	No	No	Yes	See *z/OS MVS Programming: Resource Recovery* at http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi.
Displaying units of work for CICS	Yes	No	Yes	See *CICS Operations and Utilities Guide* at http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi

Table 1. Application Server operations tasks (continued)

Task	MVS console	Application Server administrative console	TSO panel	Reference to associated procedure
Displaying units of work (transactions) for IMS	Yes	No	No	See *IMS/ESA Summary of Operator Commands* at hhttp://www.elink.ibmlink.ibm.com/public /applications/publications/cgibin/pbi.cgi.
Displaying the status of a cluster	Yes	Yes	No	See ″Example: Displaying status of clusters″ in the information center.
Displaying the status of a server	Yes	Yes	No	See ″Example: Displaying status of a server″ in the information center.
Displaying active address spaces	Yes	No	No	See ″Example: Displaying active address spaces″ in the information center.
Displaying active replies	Yes	No	No	See ″Example: Displaying active replies″ in the information center.
Modify operations				
Getting help for the modify command	Yes	No	No	See ″Example: Getting help for the modify command″ in the information center.
Canceling application clusters and servers	Yes	No	No	See ″Example: Canceling application clusters and servers with the modify command″ in the information center.
Modifying the Java trace string	Yes	No	No	See ″Example: Modifying the Java trace string″ in the information center.
Displaying status	Yes	No	No	See ″Modify command″ in the information center.
Other Application Server operations				
ARM and restart	Yes	No	No	See ″Automatic restart management″ in the information center.
Setting up error log streams for different clusters and servants	No	You can associate a log stream with a cluster from the administrative console	No	See ″Setting up the error log″
Setting up System Management Facilities recording	Yes	Enable it from here, but initiate it from the MVS console.	No	See ″Collecting job-related information with Systems Management Facility (SMF)″
Shutting down the WebSphere Application Server for z/OS environment	Yes	Application server only	No	See ″Stopping clusters″, Stopping servers″, and Steps for stopping or canceling the location service daemon from the MVS console in the information center.

Table 1. Application Server operations tasks (continued)

Task	MVS console	Application Server administrative console	TSO panel	Reference to associated procedure
Taking a WebSphere Application Server for z/OS system cluster out of service	Yes	Application server only; You cannot take a WebSphere Application Server for z/OS system cluster out of service from the administrative console	No	
Workload Management				
Displaying the status of a WLM application environment	Yes	No	No	See ″Handling workload management and server failures″ in the information center.
Handling workload management and server failures	Yes	No	No	See ″Handling workload management and server failures″ in the information center.
Getting out of the stopped state and back to the available state for an application environment	Yes	No	No	See ″Handling workload management and server failures″ in the information center.
Checking and managing the workload management application environment (display, stop/quiesce, restart/resume)	Yes	No	No	See ″Handling workload management and server failures″ and ″WLM dynamic application environment operator commands″ in the information center.

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.
- ″Clustering servers″ Includes topics about clusters, such as creating clusters, creating cluster members, querying a cluster state, removing clusters, and so on.
- Configuring security Includes security tasks, for example, enabling and disabling global 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.
- ″Dynamic caching″ Includes caching topics, for example, creating, viewing and modifying a cache instance.
- ″Troubleshooting with scripting″ 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. For additional information, see ″Using command line tools″ in the information center. Using the tools, you can start and stop application servers, check server status, add or remove nodes, and complete similar tasks.

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

Introduction: Administrative programs

The product supports a Java programming interface for developing administrative programs. See ″Using administrative programs (JMX)″ in the information center. 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

Administrative tasks typically involve defining new configurations of the product or performing operations on managed resources within the environment. IBM WebSphere Application Server configuration data is kept in files. Because all product configuration involves changing the content of those files, it is useful to know the structure and content of the configuration files. See ″Configuration documents″ in the information center.

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.

Welcome to basic administrative architecture

This article discusses basic concepts in the administrative architecture to help you understand system administration in a WebSphere Application Server environment. The fundamental concepts for WebSphere Application Server administration include software processes called servers, topological units referenced as nodes and cells, and the configuration repository used for storing configuration information.

Servers perform the actual running of the code. Several types of servers exist depending on the configuration. Each server runs in its own Java virtual machine (JVM). The application server is the primary run-time component in all WebSphere Application Server configurations. All WebSphere Application Server configurations can have one or more application servers. In some configurations, each application server functions as a separate entity. No workload distribution or common administration among application servers exists. In other configurations, workload can be distributed between servers and administration can be done from a central point.

A node is a logical group of WebSphere Application Server-managed server processes that share a common configuration repository. A node is associated with a single WebSphere Application Server profile. A WebSphere Application Server node does not necessarily have a one-to-one association with a system. One computer can host arbitrarily many nodes, but a node cannot span multiple computer systems. A node can contain zero or more application servers.

The configuration repository holds copies of the individual component configuration documents that define the configuration of a WebSphere Application Server environment. All configuration information is stored in .xml files.

A cell is a grouping of nodes into a single administrative domain. A cell can consist of multiple nodes, all administered from a deployment manager server. When a node becomes part of a cell (a federated node), a node agent server is installed on the node to work with the deployment manager server to manage the WebSphere Application Server environment on that node.

When a node is a standalone node, not part of a cell, the configuration repository is fully contained on the node. When a node is part of a cell, the configuration and application files for all nodes in the cell are centralized into a cell master configuration repository. This centralized repository is managed by the deployment manager server and synchronized to local copies that are held on each node. The local copy of the repository that is given to each node contains just the configuration information needed by that node, not the full configuration that is maintained by the deployment manager.

WebSphere Application Server types

This section discusses the three server types that interact to perform system administration.

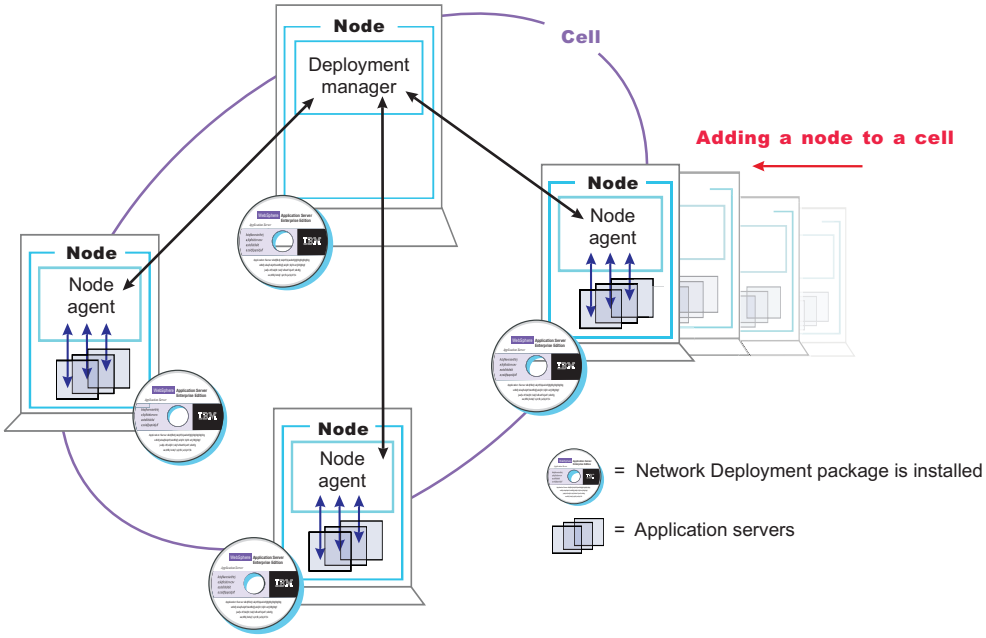
Application Server: A WebSphere Application Server provides the functions that are required to support and host user applications. An application server runs on only one node, but one node can support many application servers.

Node agent: When a node is federated, a node agent is created and installed on that node. The node agent works with the deployment manager to perform administrative activities on the node.

Deployment manager: With the deployment manager, you can administer multiple application servers from one centralized manager. The deployment manager works with the node agent on each node to manage all the servers in a distributed topology.

The following diagram depicts the concepts that are discussed in this article.

IBM WebSphere Application Server Network Deployment package



The concepts that are discussed in this article form the basis of WebSphere Application Server administration. More detailed descriptions can be found in other sections.

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” on page 11.

Clusters

Workload management optimizes the distribution of client processing tasks. Incoming work requests are distributed to the application servers that can most effectively process the requests. Workload management also provides failover when servers are not available, improving application availability.

Clusters are sets of application servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines, as opposed to the servers that are part of the same node and must be located on the same host machine.

For additional overview, refer to “Introduction: Clusters” on page 12.

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″ in the information center.

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″ in the information center for more information.
- Plug into an application server to define a hook point that runs when the server starts and shuts down. See ″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 ″Using the JVM″ in the information center for more information.
- Use an Object Request Broker (ORB) for RMI/IIOP communication. See “Managing Object Request Brokers” on page 1289 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.

In a deployment manager cell, there can be WebSphere Application Server version 5 nodes. If a version 5 node is configured to use V5 default messaging (the version 5 embedded messaging), there can be at most one JMS server on that node.

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.

Introduction: Web servers

The application server and Web server communicate using Web server plug-ins. ″Communicating with Web servers″ escribes 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 a previously defined managed or unmanaged n6de. After you define the Web server to a node, you can use the administrative console to perform the following functions for that Web server.

If the Web server is defined to a managed node, you can:

- Check the status of the Web server
- Generate a plug-in configuration file for that Web server.
- Propagate the plug-in configuration file after it is generated.

If the Web server it is defined to an unmanaged node, you can:
- Check the status of the Web server
- Generate a plug-in configuration file for 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″ in the information center.

Introduction: Clusters

Clusters are groups of servers that are managed together and participate in workload management. A cluster can contain nodes or individual application servers. A node is usually a physical computer system with a distinct host IP address that is running one or more application servers. Clusters can be grouped under the configuration of a cell, which logically associates many servers and clusters with different configurations and applications with one another depending on the discretion of the administrator and what makes sense in their organizational environments.

Clusters are responsible for balancing workload among servers. Servers that are a part of a cluster are called cluster *members*. When you install an application on a cluster, the application is automatically installed on each cluster member.

Node groups bound clusters. All cluster members of a given cluster must be members of the same node group. For more information about clusters and node groups, see ″Clusters and node groups″ in the information center.

To learn more about clusters, see ″Clusters and workload management″ and ″Balancing workloads with clusters″ for more information.

Core groups

A group of clusters can be defined as a *core group*. All of the application servers defined as a member of one of the clusters included in a core group are automatically members of that core group. Individual application servers that are not members of a cluster can also be defined as a member of a core group. The use of core groups enables WebSphere Application Server to provide high availability for applications that must always be available to end users. You can also configure core groups to communicate with each other using the *core group bridge*. The core groups can communicate within the same cell or across cells.

To learn more about core groups, see ″Setting up a high availability environment″ in the information center.

Introduction: Environment

The environment of the product applies to the configuring of Web server plug-ins, variables, and objects that you want consistent throughout a cell.

Cell-wide settings

Cell-wide settings are sets of configuration data that are stored in files in the cell directory. These configuration files are replicated to every node in the cell. Several different configuration settings apply to the entire cell. These settings include the definition of virtual hosts, shared libraries, and any variables that must be consistent throughout the entire cell.

For more information, refer to “Introduction: Cell-wide settings.”

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 ″Setting up a high availability environment″ in the information center.

Introduction: Cell-wide settings

The configuration data for WebSphere Application Server is stored in XML files. The XML files exist in one of several directories in the configuration repository tree.

The directory in which a configuration file exists determines its scope, or how broadly or narrowly that data applies. Files in an individual server directory apply to that specific server only. Files in a node-level directory apply to every server on that node. Files in the cell directory apply to every server on every node within the entire cell.

Cell-wide settings are configuration files in the cell directory. The files are replicated to every node in the cell. Several different configuration settings apply to the entire cell. These settings include the definition of virtual hosts, shared libraries, and any variables that you want consistent throughout the entire cell.

Introduction: Variables

Variables in the WebSphere environment come in many varieties. Variables 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 custom properties, environment variables, and WebSphere-specific variables.

Environment variables

Environment variables, also called *native environment variables*, are not specific to the 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.

Environment variables are specified in the administrative console. Click **Application Server >***server_name***> Process Definition > Servant Process > Environment Entries**.

This path is also used to set environment variables that control the collection of application server and Web container information in z/OS System Management Facility (SMF) records.

WebSphere variables

WebSphere 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.
- Configuring the WebSphere Application Server for z/OS location service.

WebSphere variables are specified in the administrative console by clicking **Environment > Manage WebSphere variables**. How the WebSphere variable is set determines its scope. A variable can apply to a cell, 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.
- At the cell level, it applies to all nodes in that cell, unless you set the same variable at the node or server level.
  - If you set the same variable at the server level, for that server, the setting that is specified at the server level overrides the setting that is specified at the cell level.
  - If you set the same variable at the node level, for all servers in that node, the setting that is specified at the node level overrides the setting that is specified at the cell 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 HTTP transport custom properties, follow one of the following paths:

- **Servers > Application Servers >** *server_name* **> Web Container > HTTP Transport > Additional Properties > Custom Properties**
- **Servers > Application Servers >** *server_name* **> Web Container > Additional Properties > 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 the HTTP transport custom properties page apply only to that specific transport. If the same properties are set on both pages, the settings on the transport page override the settings that are defined on the Web container page for that specific transport.

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

- Establish the application serving environment
- Secure the application serving environment - see Security
- Set up Web access for applications
- Set up resources for applications to use
- Configure class loaders - see development and deployment
- Deploy and administer applications
- Use the administrative clients
- Troubleshoot deployment and administration

Legend for ″How do I?...″ links

Documentation	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+

Establish the application serving environment

The following tasks involve establishing application serving capability in your network environment, whether you use single or clustered application servers. Servers can be grouped into administrative domains known as nodes and cells.

See also the overview:

- Version 6 topology and terminology

Administer nodes

A node is a grouping of managed servers. Use this task to view information about and manage nodes.

Documentation: ″Managing nodes″

Show me

Tell me:

- Add and remove nodes
- Manage node groups
- Cell, deployment managers, nodes, and node agents

Administer node agents

Node agents are administrative agents that represent a node to your system and manage the servers on that node. Node agents monitor application servers on a host system and route administrative requests to servers. A node agent is created automatically when a node is added to a cell.

Documentation: See	Show me	Tell me
″Managing node agents″ in the information center.		

Administer cells

When you installed the WebSphere Application Server Network Deployment product, a cell was created. A cell provides a way to group one or more nodes of your Network Deployment product. You probably will not need to reconfigure the cell. Use this task to view information about and manage a cell.

Documentation: See	Show me	Tell me
″Configuring cells″ in the information center.		

Administer configurations

Application server configuration files define the available application servers, their configurations, and their contents. You should periodically save changes to your administrative configuration. You can change the default locations of configuration files, as needed.

Documentation: See	Tell me:
″Working with server configuration files″ in the information center.	• Repository • Archives

---

Configure remote file services

Configuration data for the WebSphere Application Server product resides in files. Two services help you reconfigure and otherwise manage these files: the file transfer service and file synchronization service. By default, the file transfer service is always configured and enabled at a node agent, so you do not need to take additional steps to configure this service. However, you might need to configure the file synchronization service.

Documentation:	Tell me
″Configuring remote file services″	

Administer application servers

Create, configure, and operate application server processes. An application server configuration provides settings that control how an application server provides services for running enterprise applications and their components.

Documentation:

- ″Administering application servers″
- ″Configuring servers with scripting″
- ″Managing servers with scripting″

Show me

Tell me

Administer other server types

One step in the process of creating an application server is to specify a template. A server template is used to define the configuration settings of the new server. You have the option of specifying the default server template or choosing a template that is based on a server that already exists. The default template will be used if you do not specify a different template when you create the server.

You can create other types of servers, to represent Web servers in your topology, or for other purposes. There are two types of *generic* servers: (1) Non-Java applications or processes, or (2) Java applications or processes. A *custom service* provides the ability to plug into a WebSphere application server to define a hook point that runs when the server starts and shuts down.

Documentation:

- ″Creating generic servers″
- ″Custom services″

Tell me:

- Generic servers

Guide me (Web servers)

Balance workloads by clustering application servers

To monitor application servers and manage the workloads of servers, use server clusters and cluster members provided by the Network Deployment product.

Documentation:

- ″Balancing workloads with clusters″
- ″Clustering servers with scripting″

Show me

Tell me:

- WLM details
- Data replication service

Establishing high availability (HA) for failover

Planning ahead for high availability support is important in order to avoid the risk of a failure without failover coverage. The application server runtime of the infrastructure managed by a high availability manager includes such entities as cells and clusters. These components relate closely to core groups, high availability groups, and the policy that defines the high availability infrastructure. In a properly configured high availability environment, a high availability manager can reassess the environment it is managing and accept new components as they are added to the environment.

Documentation:	Tell me:
″Setting up a high availability environment″	• Overview • Details, core groups

---

Administer the UDDI registry

The UDDI Registry is supplied as a J2EE application file, uddi.ear. Change its configuration properties using the assembly tools. You can use either the WebSphere Application Server administrative console or the Java Management Extensions (JMX) management interface to manage UDDI Registries.

Documentation:	Tell me
• Configure • Administer	

Set up Web access for applications

These tasks involve enabling HTTP requests for applications on the application server.

Administer communication with Web servers (plug-ins)

The product provides plug-ins for supported Web servers, to enable the Web servers to pass requests to the application server, for applications running on the application server. See also the Web server related tasks in ″How do I install an application serving environment?″.

Documentation:	Show me	Tell me	Guide me ″Cheat sheets for the administrative console″
• Console: ″Communicating with Web servers″ • Scripting: ″Configuring connections to Webservers with scripting″			

Administer HTTP sessions

Configure the service that the product provides for managing HTTP sessions: Session Manager.

Documentation:	Show me
• Console • Scripting: ″Configuring applications for session management using scripting	

Administer IBM HTTP Server Version 6.x

The product provides a complementary Web server with its own documentation that can be installed into the information center.

Set up resources for applications to use

Make a variety of resources available to your applications that are deployed on the application server.

Provide access to naming and directory resources (JNDI)

Configure naming. Naming is used by clients of WebSphere Application Server applications to obtain references to objects related to those applications, such as Enterprise JavaBeans (EJB) homes. These objects are bound into a mostly hierarchical structure, referred to as a 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.

Documentation:
- Name server
- Bindings

Tell me:
- Introduction
- Basic concepts
- Advanced concepts
- Examples
- Troubleshooting

Provide access to relational databases (JDBC resources)

Configure data sources that applications use to access the data from databases.

Documentation:	Show me:	Tell me	Guide me
• Console	• Cloudscape		
• Scripting	• DB2		
	• Oracle		

Provide access to messaging resources (default messaging provider)

Use one of various ways to implement a messaging provider for use with WebSphere Application Server. A messaging provider enables use of the Java Messaging Service (JMS) and other message resources in the product.

Documentation:	Show me	Tell me
• Console		
• Scripting		

Use IBM service integration technologies

Tell me:

- Overview
- Architecture
- Mediation

Establish workload balancing and high availability (HA) of messaging engines

Tell me

Access Service Integration (SI) bus resources

Show me	Tell me:
	• Service integration bus resources • JMS resources for service integration bus

---

Deploy and administer applications

These tasks involve deploying applications onto the application server, then administering the applications.

Install applications

Installable modules include enterprise archive (EAR), enterprise bean (EJB), Web archive (WAR), resource adapter (connector or RAR), and application client files.

Documentation	Show me	Tell me
• Console:Installing application files • Scripting: ″Deploying applications using scripting″		

Start and stop 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).

Documentation:	Show me	Tell me
• Starting and stopping applications • Starting applications with scripting		

Update applications

Update deployed applications or modules using the administrative console or **wsadmin** scripting. Learn which changes are candidates for hot deployment and dynamic reloading, in which you can make various changes to applications and their modules without having to stop the server and start it again.

Documentation:	Tell me	Teach me
• Updating applications • ″Updating installed applications with the wsadmin tool″		

Deploy applications rapidly (WebSphere Rapid Deployment)

Take advantage of new rapid deployment capabilities. 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. Refer to the **Rapid deployment tools** documentation in the information center.

Enhanced EAR files

	Tell me	Teach me

Deploy and administer Web services applications

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 has been configured and enabled for Web services. You can use either the administrative console or the wsadmin scripting interface to deploy an EAR file.

Documentation	Show me	Tell me

Use the administrative clients

A variety of tools are provided for administering the product.

Choose an administrative client

Learn about and decide among the available administrative clients, including a graphical console, scripting (wsadmin), command line tools, and Java Management Extensions (JMX) programs.

Documentation:	Tell me
″Using the administrative clients″	

Use the administrative console

The administrative console is a Web-based tool that you use to administer the product. The administrative console supports a full range of product administrative activities.

Documentation	Show me	Tell me

Use scripting (wsadmin)

Scripting is a non-graphical alternative that you can use to configure and manage WebSphere Application Server. The WebSphere Application Server **wsadmin** tool provides the ability to run scripts. The tool supports a full range of product administrative activities.

″Using scripting (wsadmin)″		Tell me

See also:

- Start, stop, monitor processes
- Other administrative commands

Troubleshoot deployment and administration

Troubleshoot problems that occur when you are deploying applications onto the application server, or when you are administering an established application serving environment.

Troubleshoot administration

Review some possible causes, based on the error you are seeing.

Documentation

Chapter 3. Starting and stopping quick reference

This topic describes how to start and stop the main operations in your application serving environment. It also provides a quick guide to accessing the main 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 be in the `install_root/profiles/profile_name/bin`directory to start and stop the server.
Deployment manager Go to the *install_root* `/bin`directory of a Network Deployment installation and run the following command. See ″startManager command″ in the information center for details and variations. `startManager`
Application server Go to the *install_root* /bin directory of a WebSphere Application Server or Network Deployment installation and run the following command. See ″startServer command″ in the information center for details and variations `startServer server` where *server* is the application server that you want to start. Issuing START *appserver_proc_name* and STOP *appserver_proc_name* in the Using the MVS console is an additional option on z/OS systems.
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` To start and stop application server clusters, see ″Starting clusters″.

- Use administrative clients and tools.

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

  - You also can :Use the MVS console″ on z/OS systems.
  - To launch a scripting client, see ″Starting the wsadmin scripting client″ in the information center.
  - 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 1674.

  See the administrator commands that are listed in the **Reference** section of the information center.
- Use development and deployment tools. Use the following tools 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. The tools are available for use on distributed operating systems.

  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 ″Starting an assembly tool″ in the information center.

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 244.

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.

  Customization dialog
  After ″Installing the product and additional software″, use the customization dialog to configure the product. See ″Starting the Customization Dialog″ in the information center.
- Use troubleshooting tools - see ″Working with diagnostic tools and controls″ in the information center.

Chapter 4. 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 run-time behavior of packaged applications of deployed applications.

This article 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 article “Class loading: Resources for learning” on page 35 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 (See ″Introduction: Mail, URLs, and other J2EE resources″ in the information center) 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 shard library for each library file that your applications need. See ″Creating a shared library″ in the information center
   b. Define a library reference instance for each shared library that your application uses. See, ″Associating shared libraries with applications″ in the information center.
6. **Optional:** Configure class preloading.

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.

The run-time 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. See ″Managing shared libraries″ in the information center.

2. A WebSphere-specific extensions class loader

   The WebSphere Application Server 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 Application Server 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 adapters 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-specific 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 Application Server 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 preloading

The first time that a WebSphere Application Server process starts, the name of each run-time class that is loaded and the name of the JAR file that contains the class are written to a preload file. The names of non-runtime classes such as custom services, resource classes such as `db2java.zip`, classes on the JVM class path, and application classes are not written to the preload file. Subsequent startups of the process use the preload file to start the process more quickly.

Preload files have the `.preload` extension. WebSphere Application Server processes that have preload files include the following:

Process	Preload file name
Application server	*cell_name.node_name.server_name*.preload
`startserver`	`WsServerLauncher.preload`
`launchClient`	`launchClient.preload`

Running the `startserver server1` command causes the `startserver` command to use a `WsServerLauncher.preload` file and the server to use a *cell_name.node_name*`.server1.preload` file. Later, running a command such as `startserver server1 -script`, where the -script option creates a new script, uses the *cell_name.node_name*`.server1.preload` file only.

Preload files, by default, are installed in the *install_root* directory.

New classes required during the startup of a process are added to the preload file. Any classes that are removed from a process are ignored during subsequent startups. Although it is not necessary, an administrator can delete the preload file and force a refresh that removes the ignored classes from the file.

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:

Application class-loader policy

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 class-loader policy

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.

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

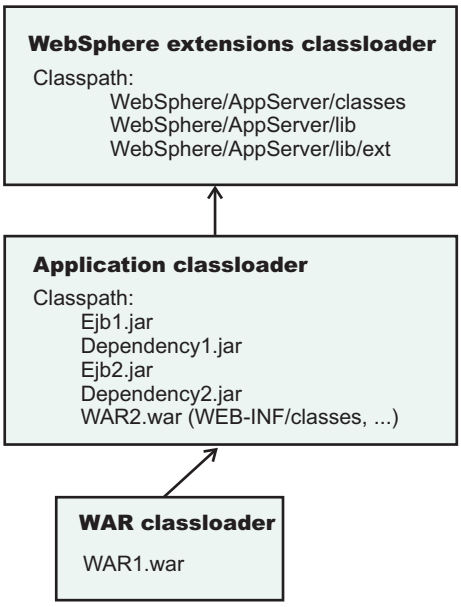
For each application server in the system, you can set the application class-loader policy to `Single` or `Multiple`. When the application class-loader policy is set to `Single`, then a single application class loader loads all EJB modules, dependency JAR files, and shared libraries in the system. When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

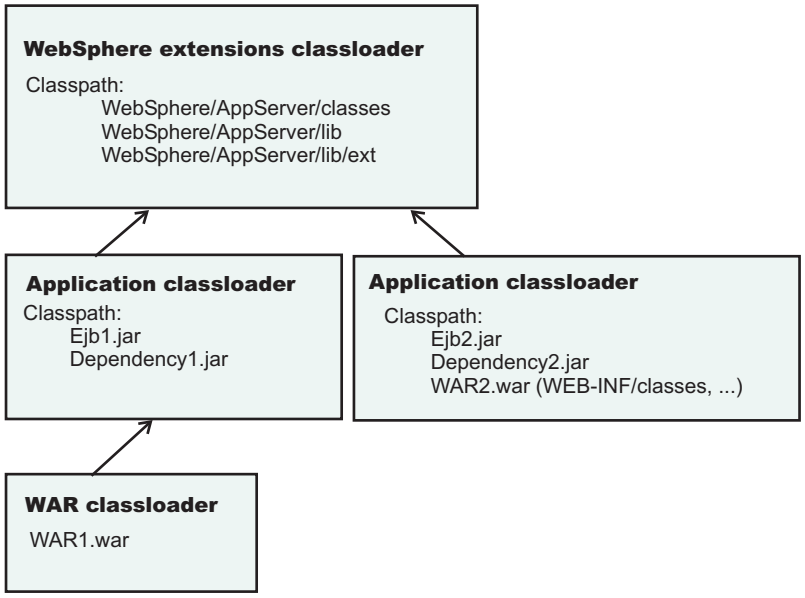
```
Server's application class-loader policy: Single
Application's WAR class-loader policy: Module

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

Two values for a class-loader mode are supported:

Parent First

The `Parent First` class-loader mode causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path. This value is the default for the class-loader policy and for standard JVM class loaders.

Parent Last

The `Parent Last` class-loader mode causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent. Using this policy, an application class loader can override and provide its own version of a class that exists in the parent class loader.

The following settings determine the mode of a class loader:

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

Configuring class loaders of a server

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

This article assumes that an administrator created an application server on a WebSphere Application Server product. See ″Creating application servers″.

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. See ″Application server settings″ for more information.

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: See ″Application server settings″ in the information center.

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 mode. The `Parent 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 `Parent Last` value causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent.
   d. Click **OK**.

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

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 mode

Specifies the class-loader mode when the application class-loader policy for the application server is `Single`.

You specify a policy of `Single` on the settings page for the application server, accessed by clicking **Servers > Application Servers >***server_name*. When the policy is `Single`, applications are not isolated from each other. A single application class loader contains all of the EJB modules, dependency JAR files, and shared libraries in the system.

The `Parent 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 `Parent Last` value causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent. Specifying the `Parent Last` value enables an application class loader to override and provide its own version of a class that exists in the parent class loader.

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 mode

Specifies the class-loader mode when the application class-loader policy for the application server is `Single`.

You specify a policy of `Single` on the settings page for the application server, accessed by clicking **Servers > Application Servers >***server_name*. When the policy is `Single`, applications are not isolated from each other. A single application class loader contains all of the EJB modules, dependency JAR files, and shared libraries in the system.

The `Parent 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 `Parent Last` value causes the class loader to attempt to load classes from its local class path before delegating the class loading to its parent. Specifying the `Parent Last` value enables an application class

loader to override and provide its own version of a class that exists in the parent class loader.

Data type	String
Default	Parent First

Configuring application class loaders

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

This article 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* to access the settings page for an enterprise application.
2. Specify the class-loader mode for the application. The application class-loader mode 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 mode**:

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

3. 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 `Module`, which uses a separate class loader to load each WAR file. Setting the value to `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
Module	Uses a different class loader for each WAR file.
Application	Uses a single class loader to load all of the WAR files in your application.

4. Specify whether to enable class reloading when application files are updated. By default, class reloading is not enabled. See the description for **Enable class reloading** in “Enterprise application settings” on page 56 for details on enabling and disabling reloading. You might specify different values for EJB modules and for Web modules such as servlets and JavaServer page (JSP) files.
5. Specify the number of seconds to scan the application’s file system for updated files. The value specified for **Reloading interval** 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).
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 article assumes that you installed a Web module on an application server.

Configure the class-loader mode 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 `Module`. If the policy is set to `Module`, then each Web module receives its own class loader whose parent is the application class loader. If the policy is set to `Application` on the settings page of an enterprise application, 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 **WAR class loader policy**, **Enable class reloading**, and **Reloading interval** can affect Web module class loading. If **WAR class loader policy** is set to `Module`, 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 `Application`, then the application class loader loads the Web module contents.
2. Click **Applications > Enterprise Application >***application_name* **> Web modules >***Web_module_name* to access the settings page for a deployed Web module.
3. Specify the class-loader mode 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 mode**:

Option	Description
Parent 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 `Parent last`. 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 `Parent first` is specified, the WAR class loader never receives a request to load a class.
Parent last	Causes the class loader to search in the WAR class loader first to load a class. By specifying `Parent last`, your WAR class loader can override classes contained in the parent application class loader. **Tip:** Specifying the `Parent last` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

4. Click **OK**.

Save the changes to the administrative configuration.

Configuring class preloading

Class preloading affects how quickly a WebSphere Application Server process starts.

The first time that a WebSphere Application Server process starts up, the name of each class that is loaded and the name of the JAR file that contains the class are written to a preload file. Subsequent startups of the process use the preload file to start the process more quickly.

No configuring of class preloading is necessary.

However, an administrator can disable or enable preloading explicitly. By default, class preloading is enabled for WebSphere Application Server processes. To change the configuration for class preloading, an administrator sets new values for system properties.

- Disable class preloading. Set the Java virtual machine (JVM) system property ibm.websphere.preload.classes to `false`.
  1. In the administrative console, click **Servers > Application Servers >***server_name* **> Java and Process Management > Process Definition > Java Virtual Machine** to access the Java Virtual Machine page. See ″Java virtual machine settings″ in the information center
  2. On the Java Virtual Machine page, specify `-Dibm.websphere.preload.classes=false` for **Generic JVM arguments**.
  3. Click **OK**.
  4. Save your administrative configuration.

5. Stop the application server and then restart the application server. See ″Stopping servers″ and ″Starting servers″ in the information center.

- Enable class preloading again. If you disabled class preloading, you can enable it again by doing either of the following:
  - Set the JVM system property to `true`. On the Java Virtual Machine page, specify `-Dibm.websphere.preload.classes=true` for **Generic JVM arguments**. See ″Java virtual machine settings″ in the information center.
  - Remove the JVM system property that was created to disable class preloading. On the Java Virtual Machine page, remove the value `-Dibm.websphere.preload.classes=false` specified for **Generic JVM arguments**. See ″Java virtual machine settings″ in the information center.

  After you change the JVM system property, click **OK**, save your administrative configuration, stop the application server, and then restart the application server. See ″Stopping servers″ and ″Starting servers″ in the information center.
- Regenerate a class preload file. Delete the `.preload` file for the WebSphere Application Server process. When the process next starts up, a new class preload file is generated for the process.

Class loading: Resources for learning

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
- J2EE Class Loading Demystified
- Understanding J2EE Application Server Class Loading Architectures

Programming instructions and examples
- Developing and Deploying Modular J2EE Applications with WebSphere Studio Application Developer and WebSphere Application Server
- IBM WebSphere Application Server Programming

Programming specifications
- Sun's J2EE™ Platform Specification
- Sun's J2EE™ Extension Mechanism Architecture

Chapter 5. 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 and assemble the module and configure the target server. Before choosing a server as a target for the module, ensure that the node version for the server is compatible with your module. See ″Assembling applications″ and ″Administering application servers″ in the information center.

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.

- Install application files on an application server.
- Edit the administrative configuration for an application.
- 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.

If a changed application or module is deployed on a cluster, click **Rollout Update** on the Enterprise Applications page to propagate the changed configuration on all cluster members of the cluster on which the application or module is deployed. **Rollout Update** sequentially updates the configuration on the nodes that contain cluster members.

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
- Zero or more Web modules
- Zero or more connector modules (packaged in RAR 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 *adminconsole* 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. Users cannot change the metadata for a system application such as its J2EE bindings or J2EE extensions, unless the metadata assigns users and groups for security purposes. Non-security related metadata requiring a change 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), and resource adapter (connector or RAR) files. 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 - See ″Getting started with scripting″ in the information center.
   - Java administrative programs - See ″Using administrative programs (JMX)″ 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.

When saving the configuration, synchronize the configuration with the nodes where the application is expected to run.

Next, 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.

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 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 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, calls a 6.x API or uses a 6.x feature, then you must install the module on a 6.x deployment target.

Selecting options such as **Pre-compile JSP**, **Use Binary Configuration**, **Deploy Web services** or **Deploy enterprise beans** during application installation to a 6.x server or a 6.x deployment manager indicates that the application uses 6.x product features. You cannot deploy such applications on a 5.x deployment target. You must deploy such applications on a 6.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 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.

Table 2. 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
Application, EJB, Web, or client	J2EE 1.3	Yes	6.x	6.x
Application, EJB, 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
Resource adapter	JCA 1.5	Yes or No	6.x	6.x

Ways to install applications or modules

WebSphere Application Server provides several ways to install application files on a server or cluster.

Installable files include enterprise archive (EAR), enterprise bean (EJB), Web archive (WAR), resource adapter (connector or RAR), and application client modules.

Table 3. 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 41.	Click **Applications > Install New Application** in the console navigation tree and follow instructions in the wizard.	All EAR, EJB, WAR, RAR, and application client files	Provides one of the easier ways to install application files. See “Preparing for application installation settings” on page 46 for guidance. For applications that do not require changes to the default bindings, select the **Generate Default Bindings** option and then, on the Summary panel, 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. See ″Installing applications with the wsadmin tool″ in the information center.	All EAR, EJB, WAR, 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. See ″Starting an application through programming″. .
Java application programming interfaces	Install programs by completing the steps in ″Managing applications through programming″.	All EAR files	Use MBeans to install the application.	Start the application by calling the *startApplication* method on a proxy.

Table 3. Ways to install application files (continued)

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, 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.
Java programs	Code programs that use J2EE DeploymentManager (JSR-88) methods.	All J2EE modules, including EAR files and standalone EJB, WAR, 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, or Java programs that call J2EE DeploymentManager (JSR-88) methods. This article describes how to use the administrative console to install an application, EJB component, 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. The first of two Preparing for application installation pages is displayed.
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) or Java archive (JAR) file for installation.
   b. If you are installing a standalone WAR file, specify the context root.

c. Click **Next**.

3. On 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 46 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 (last step below). "Example: Installing an EAR file using the default bindings" on page 51 provides sample steps.
4. On the **Step: Select installation options** panel, provide values for the following settings specific to WebSphere Application Server. Default values are used if you do not specify a value.
   a. For **Pre-compile JSP**, specify whether to precompile JavaServer page (JSP) files as a part of installation. The default is not to precompile JSP files. Install onto a 6.x deployment target. If you select **Pre-compile JSP** and try installing your application onto a 5.x deployment target, the installation is rejected. For this option, install only onto a 6.x deployment target.
   b. For **Directory to install application**, specify the directory to which the 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 *install_root*/installedApps; for example, C:\WebSphere\AppServer\profiles*profile _name*\installedApps*cell_name*.

      Note: If an installation directory is not specified when an application is installed on a single-server (base) configuration, the application is installed in APP_INSTALL_ROOT/*base_cell_name*. When the base server is made a part of a Network Deployment configuration (using the addNode utility), the cell name of the new configuration becomes the cell name of the deployment manager node. If the -includeapps option is used for the addNode utility, then the applications that are installed prior to the addNode operation still use the installation directory APP_INSTALL_ROOT/*base_cell_name*. However, an application that is installed after the base server is added to the network configuration uses the default installation directory APP_INSTALL_ROOT/*network_cell_name*. To move the application to the APP_INSTALL_ROOT/*network_cell_name* location upon running the addNode operation, you should explicitly specify the installation directory as ${APP_INSTALL_ROOT}/${CELL} during installation. In such a case, the application files can always be found under APP_INSTALL_ROOT/*current_cell_name*.
   c. For **Distribute application**, specify whether WebSphere Application Server expands or deletes application binaries in the installation destination. The default is to enable application distribution. As a result, when you save changes in the console, application binaries for newly installed applications are expanded to the directory specified. The binaries are also deleted when you uninstall and save changes to the configuration. 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 is expected to run.

      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.
   d. For **Use Binary Configuration**, specify 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 EAR file. The default is not to use the binary configuration. If you select **Use Binary Configuration**, your application files must be installed onto a 6.x deployment target. The files cannot be installed onto a 5.x deployment target.

e. For **Deploy enterprise beans**, specify whether the EJBDeploy tool runs during application installation. The tool generates code needed to run EJB files. You must enable this setting in the following situations:
   - The EAR file was assembled using an assembly tool such as 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.
   - The EAR file was assembled using versions of the Application Assembly Tool (AAT) previous to Version 5.

   Enabling this setting might cause the installation program to run for several minutes. Also, install onto a 6.x deployment target. If you select **Deploy enterprise beans** and try installing your application onto a 5.x deployment target, the installation is rejected. For this option, install only onto a 6.x deployment target.

f. For **Application name**, name the application. Application names must be unique within a cell and cannot contain characters that are not allowed in object names. See ″Object names″.

g. For **Create MBeans for resources**, specify whether to create MBeans for various resources (such as servlets or JSP files) within an application when the application is started. The default is to create MBean instances.

h. For **Enable class reloading**, specify whether to enable class reloading when application files are updated. The default is not to enable class reloading. For EJB modules or any non-Web modules, enabling class reloading sets 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 Web modules such as servlets and 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`. You can set reloadingEnabled to `true` when editing the extended deployment descriptors of your Web module in an assembly tool.

   To disable reloading of a Web module, set the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file to `false`. Or, if the Web module has the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file set to `true`, enable class loading, and set the **Reload interval** property to zero (0).

i. For **Reload interval in seconds**, specify the number of seconds to scan the application's file system for updated files. The default is the value of the reload 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 reload interval specified here takes effect only if class reloading is enabled.

j. For **Deploy Web services**, specify 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. Note that if you select **Deploy** and try installing your application onto a 5.x deployment target, the installation is rejected. For this option, install only onto a 6.x deployment target.

k. For **Validate Input off/warn/fail,** specify 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.

l. For **Process embedded configuration**, specify 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`.

5. On the **Step: Map modules to servers** panel, for every module select a target server or cluster from the **Clusters and Servers** list. Select the check box beside **Module** to select all of the application modules or select individual modules. Ensure that you are installing your application onto an appropriate deployment target. You can specify Web servers as targets that route requests to the application. The plug-in configuration file `plugin-cfg.xml` for each Web server is generated based on the applications which are routed through it. If you want a Web server to serve the application, use the **Ctrl** key to select an application server or cluster and the Web server together in order to have the plug-in configuration file `plugin-cfg.xml` for that Web server generated based on the applications which are routed through it.
6. If your application uses EJB modules, on the **Step: 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`.
7. If your application uses EJB modules that contain Container Managed Persistence (CMP) beans that are based on the EJB 1.x specification, for **Step: Provide default datasource mapping 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.
8. If your application has CMP beans that are based on the EJB 1.x specification, for **Step: Map datasources for all 1.x CMP**, 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.
9. If your application defines EJB references, for **Step: 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.
10. If your application defines resource references, for **Step: 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.
11. If your application uses Web modules, for **Step: 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.
12. If the application has security roles defined in its deployment descriptor then, for **Step: Map security roles to users/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 if predefined users such as **Everyone** or **All authenticated users** are mapped to it. To select specific users or groups from the user registry:
    a. Select a role and click **Lookup users** or **Lookup groups**.
    b. On the Lookup users/groups panel displayed, enter search criteria to extract a list of users or groups from the user registry.
    c. Select individual users or groups from the results displayed.

d. Click **OK** to map the selected users or groups to the role selected on the **Step: Map security roles to users/groups** panel.

13. If the application has Run As roles defined in its deployment descriptor, for **Step: Map RunAs roles to user**, 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**.
14. If your application contains EJB 1.x CMP beans that do not have method permissions defined for some of the EJB methods, for **Step: 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.
15. If your application contains message driven enterprise beans, for **Step: Provide Listener Ports or activation specification JNDI name for messaging beans**, provide a listener port name or an activation specification JNDI name for every message driven bean. 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**.
16. If your application uses EJB modules that contain CMP beans that are based on the EJB 2.x specification, for **Step: Provide default datasource mapping 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.
17. If your application has CMP beans that are based on the EJB 2.x specification, on the **Step: Map datasources for all 2.x CMP** 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.
18. 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 **Step: 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.
19. If the **Deploy enterprise beans** setting is enabled on the **Select installation options** panel, then you can specify options for the EJBDeploy tool on the **Step: 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 EJBDeploy tool. The tool is run on the EAR file during installation after you click **Finish**.
20. If your application contains resource environment references, for **Step: Map resource environment 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.
21. If your application defines **Run-As Identity** as *System Identity*, for **Step: Replace RunAs System to RunAs Roles**, 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.

22. If your application has resource references that map to resources that have an Oracle database doing backend processing, for **Step: Specify the isolation level for Oracle type provider**, 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.
23. If your application uses message driven beans, for **Step: Build message destination to administered objects**, specify the JNDI name of the J2C administered object to bind the message destination reference to the message driven beans.
24. If your application contains an embedded `.rar` file, for **Step: Map JCA resources to resources**, specify the name and JNDI name of each J2C connection factory, J2C administered object and J2C activation specification.
25. If your application contains an embedded `.rar` file, its activationSpec property has the value `Destination`, and its introspected type is `javax.jms.Destination`, for **Step: Bind J2CActivationSpec to Destination Jndi name**, specify the jndiName value for each activation bound to it.
26. If your application has EJB modules for which deployment code has been generated for multiple backend databases using an assembly tool, for **Step: Select a backend ID**, specify the backend ID representing the backend database to be used when the EJB module runs. See ″Mapping enterprise beans to database tables″ in the information center
27. 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. Click **Finish**.

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

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.

After the application file installs successfully, do the following:

1. Associate any shared libraries that the application needs to the application. See ″Managing shared libraries″ in the information center.
2. 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 *install_root*/installedApps/*cell_name* by default or the directory that you designate. For the Network Deployment installation, files are copied to remote nodes when the configuration on the deployment manager synchronizes with the configuration on individual nodes.
3. Start the application.
4. 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, update the application.

Preparing for application installation settings

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

Path:

Specifies the fully qualified path to the `.ear`, `.jar`, 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.You can browse the entire file system of a node if the node agent or deployment manager is running on that selected node. Only `.ear`, `.jar`, 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, 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 on a Windows machine might be `C:\WebSphere\AppServer\installableApps\test.ear`. If you are installing a stand-alone 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 orserver machine.

Context root:

Specifies the context root of the Web application (WAR).

This field is used only to install a stand-alone WAR 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`.

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.

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.

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.

Connection Factory Bindings:

Specifies the default data source JNDI name.

If **Default connection factory bindings** is selected, specify the JNDI name for the default data source to be used with the bindings. Also specify the resource authorization.

Virtual Host:

Specifies the virtual host for the Web module.

Specific bindings file:

Specifies a bindings file that overrides the default binding.

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>
```

Note: 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>
```

Note: 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>
```

Note: 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>
```

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. See ″Editing deployment descriptors″ in the information center. 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`.

   a. Select the **Remote file system** radio button and click **Browse**.
   b. On the Browse Remote Filesystems page, click on the name of the node that holds the `my_appl.ear` file, **C:**, **sample_apps**, **my_appl.ear**, and then **OK**.
3. Now that a value is given for **Specify path**, on the first Preparing for application installation page, click **Next**.
4. 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.
5. If application security warnings are displayed, read the warnings and click **Continue**.
6. On the Install New Application page, click on **Summary**, the last step.
7. On the Summary panel, verify the cell, node, and server onto which the application files will install.

   a. Beside the **Cell/Node/Server** option, click **Click here**.
   b. On the **Map modules to servers** 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**.

      Note that on the **Map modules to servers** 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.

   Because `my_appl.ear` does not require any additional settings to complete an installation, the Summary panel is displayed again.
8. 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 *install_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 article 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 J2EE 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 *adminconsole* 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 J2EE 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 Started*.
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 Stopped*.
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.
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.
Rollout Update	Sequentially updates an application installed on multiple cluster members across a cluster. After you update an application's files or configuration, click **Rollout Update** to install the application's updated files or configuration on all cluster members of a cluster on which the application is installed. **Rollout Update** does the following for each cluster member in sequence: 1. Saves the updated application configuration. 2. Stops all of the cluster members on one node. 3. Updates the application on the node by synchronizing the configuration. 4. Restarts the stopped cluster members. 5. Repeats steps 2 through 4 for all of the nodes that have cluster members. This action updates an application on multiple cluster members while providing continuous availability of the application.
Remove File	Deletes a file of the deployed application or module. **Remove File** deletes a file from the WebSphere Application Server configuration repository and from the file system of all nodes where the file is installed. If the application or module is deployed on a cluster, after removing a file click **Rollout Update** to roll out the changes across the entire cluster.
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.

Name

Specifies the name of the installed (or deployed) application. Application names must be unique within a cell and cannot contain characters that are not allowed in object names.

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.
	**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.
	**Not applicable**	Application does not provide information as to whether it is running.

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:

/	**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 binaries

Specifies the directory to which the 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 *`install_root`*`/installedApps`; for example, `C:\WebSphere\AppServer\profiles\`*`profile _name`*`\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 though it is particularly needed when installing an application on a cluster with members on heterogeneous nodes because, in such cases, there might not be a single way to specify an

absolute path. A WebSphere Application Server variable ${CELL} that denotes the current cell name can also be in the pathmap variable; for example, ${MY_APP}/${CELL}.

You can define WebSphere Application Server variables on the WebSphere Variables page of the administrative console, accessed by clicking **Environment > WebSphere Variables**.

Data type String
Units Full path name

Use metadata from binaries

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 metadata from binaries** 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 distribution

Specifies whether WebSphere Application Server expands or deletes application binaries in the installation destination. The default is to enable application distribution. Application binaries for installed applications are expanded to the directory specified. The binaries are also deleted when you uninstall and save changes to the configuration. 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 distribution** setting is the same as the **Distribute application** field on the application installation and update wizards.

Data type Boolean
Default true

Validation

Specifies whether WebSphere Application Server examines the application references specified during application installation or updating and, if validation is enabled, warns the users 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 **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 **Validation** setting is the same as the **Validate Input off/warn/fail** field on the application installation and update wizards.

Data type	String
Default	warn

Class loader mode

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 `Parent First`. By specifying `Parent Last`, your application can override classes contained in the parent class loader, but this action can potentially result in ClassCastException or LinkageErrors if you have mixed use of overridden classes and non-overridden classes.

The options are `Parent First` and `Parent Last`. 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	Parent First

WAR class loader policy

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

The options are `Application` and `Module`. The default is to use a separate class loader to load each WAR file.

Data type	String
Default	Module

Enable class reloading

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

For EJB modules or any non-Web modules, selecting **Enable class reloading** sets 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 Web modules such as servlets and JavaServer page (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`. You can set reloadingEnabled to `true` when editing your Web module's extended deployment descriptors in an assembly tool.

To enable reloading of a Web module, where you also want reloading of EJB and non-Web modules enabled:
1. Set the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file to `true`.
2. Select this **Enable class reloading** property.
3. Set the **Reloading interval** property to a value greater than zero (for example, 1 to 2147483647).

To enable reloading of a Web module only, and not enable reloading of EJB or non-Web modules:
1. Set the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file to `true`.
2. Set the IBM extension reload interval attribute in the `ibm-web-ext.xmi` file to a value greater than zero (for example, 1 to 2147483647).
3. Do not select this **Enable class reloading** property.

To disable reloading of a Web module, set the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file to `false`. Or, if the Web module has the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file set to `true`, to disable reloading using the administrative console:
1. Select this **Enable class reloading** property.

2. Set the **Reloading interval** property to zero (0).

Data type	Boolean
Default	false

Reloading interval

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 **Reloading interval** 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 reloading interval specified here overrides the value specified in the IBM extensions for each non-Web module in the EAR file (which in turn overrides the reload interval specified in the IBM extensions for the application in the EAR file). The reloading interval attribute takes effect only if class reloading is enabled.

The range is from 0 to 2147483647.

Data type	Integer
Units	Seconds
Default	3

Starting weight

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

Data type	Integer
Default	1
Range	0 to 2147483647

Background application

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 WebSphere Application Server 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 MBean files for various resources (such as servlets or JSP files) within an application when the application starts. The default is to create MBean files.

Data type	Boolean
Default	true

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, this article assumes 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 article describes how to change the settings of an application or module using the administrative console.

- Change the settings of the application or module on the settings page for the enterprise application.
  1. Click **Applications > Enterprise Applications >***application_name* in the console navigation tree.
  2. Change the values for settings as needed. The settings page help provides detailed information on the settings and allowed values. When you installed the application or module, you specified most, if not all, of the settings values. After installation, the settings on this page that you are likely to change include the following:

Enable class reloading and **Reloading interval**	These settings control 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 page (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`. Refer to the settings page help for detailed information on enabling or disabling class reloading.
Starting weight	If your application starts automatically when its server starts, this value controls how quickly the application starts. **Starting weight** specifies the order in which applications are started when the server starts. The application with the lowest starting weight is started first.
Background application	If your application starts automatically when its server starts, **Background application** 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.

  3. Click **OK**.
- Map each module of your application to a target server. Specify the application servers, clusters of application servers, or Web servers onto which to install modules of your application.
- 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.
     – Stateful session bean failover
     – Session management
     – Application profiles
     – Libraries or library references - See ″Library reference collection″
     – Target mappings

- Last participant support extension
- Deployment descriptors
- Publish WSDL files - See ″Publish WSDL zip files settings″
- Provide JMS and EJB endpoint URL information
- ″Provide HTTP endpoint URL information″
- Provide JNDI Names for Beans. For more information, refer to “Task overview: Using enterprise beans in applications” on page 169.
- Map resource references to resources
- Map EJB references to beans. For more information, refer to “Task overview: Using enterprise beans in applications” on page 169.
- Map data sources for all 2.x CMP beans
- Provide default data source mapping for modules containing 2.x entity beans. For more information, refer to “Creating and configuring a data source using the administrative console” on page 589.
- Map modules to servers
- Map virtual hosts for Web modules
- Web modules
- EJB modules
- Connector 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.

In the Network Deployment product, the application binaries are transferred to nodes when the configuration changes on the deployment manager synchronize with configurations for individual nodes on which the application will run.

If the application or module is deployed on a cluster and you have no more configuration changes to make, click **Rollout Update** on the Enterprise Applications page to propagate the changed configuration on all cluster members of the cluster on which the application or module is deployed. **Rollout Update** sequentially updates the configuration on the nodes that contain cluster members.

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. 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`.

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 a deployer. When assembling the application and then installing it 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 assembling an application. After defining the bindings, the deployer can install the application onto a server supported by WebSphere Application Server without selecting to 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.

  Note: 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* **> Provide JNDI Names for Beans** in the administrative console.

Data sources for entity beans

Entity beans such as container-managed persistence (CMP) beans store persistent data in data stores. See ″Developing enterprise beans″ and ″Developing data access applications″ in the information center. 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.

During and after application installation, data sources can be mapped to 2.x entity beans on the Map data sources for all 2.x CMP beans panel and on the Provide default data source mapping for modules containing 2.x entity beans panel. After installation, click **Applications > Enterprise Applications >***application_name* in the administrative console, then select **Map data sources for all 2.x CMP beans** or **Provide default data source mapping for modules containing 2.x entity beans**. 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.

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* **> Map EJB references to beans** in the administrative console.

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* **> Map resource references to resources** in the administrative console to access the Map resource references to resources 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. See ″How requests map to virtual host aliases″ in the information center. 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* **> Map virtual hosts for Web modules** 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 spec as defined in a connector module (`.rar` file) under **WebSphere Bindings** on the **Beans** page of an assembly tool EJB deployment descriptor editor. See ″Developing an enterprise application to use message-driven beans″ for additional information. 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 **Provide Listener Ports or activation specification JNDI name for messaging 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 643.

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 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.

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, cluster of application servers 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 article 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 Map modules to servers 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 Map modules to servers 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* **> Map modules to servers** in the console navigation tree. The Selecting servers - Map modules to servers 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 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.
   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.

In the Network Deployment product, the application binaries are transferred to nodes when the configuration changes on the deployment manager synchronize with configurations for individual nodes on which the application will run.

If the application or module is deployed on a cluster and you have no more configuration changes to make, click **Rollout Update** on the Enterprise Applications page to propagate the changed configuration on all cluster members of the cluster on which the application or module is deployed. **Rollout Update** sequentially updates the configuration on the nodes that contain cluster members.

Selecting servers - Map modules to servers 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. Each module must be mapped to a target server. 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* **> Map modules to servers**. This panel is the same as 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**.

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, calls a 6.x API or uses a 6.x feature, then you must install the module on a 6.x deployment target.

Module:

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

URI:

Specifies the location of the module archive contents on a file system. The location is relative to the application URL. The URI must match the URI of a ModuleRef URI in the deployment descriptor of the deployed application (`.ear` file).

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 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.

Use the Map virtual hosts for Web 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 Map virtual hosts for Web modules panel, 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* **> Map virtual hosts for Web modules** in the console navigation tree. The Selecting virtual hosts for Web modules - Map virtual hosts for Web modules panel 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.

Save changes to your administrative configuration.

In the Network Deployment product, the application binaries are transferred to nodes when the configuration changes on the deployment manager synchronize with configurations for individual nodes on which the application will run.

If the application or module is deployed on a cluster and you have no more configuration changes to make, click **Rollout Update** on the Enterprise Applications page to propagate the changed configuration on all cluster members of the cluster on which the application or module is deployed. **Rollout Update** sequentially updates the configuration on the nodes that contain cluster members.

Selecting virtual hosts - Map virtual hosts for Web modules 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* **> Map virtual hosts for Web modules**. 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`, 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.

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

Virtual host:

Specifies the name of the virtual host to which the Web module currently is 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.

Starting and 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 article 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 - See ″Stopping applications with scripting″ in the information center.
- Java programs that use ApplicationManager or AppManagement MBeans

This article 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*.
2. Specify a different value for **Starting weight**.

   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 **Background application**.

   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.

5. If the application or module is deployed on a cluster and you have no more configuration changes to make, click **Rollout Update** on the Enterprise Applications page to propagate the changed configuration on all cluster members of the cluster on which the application or module is deployed. **Rollout Update** sequentially updates the configuration on the nodes that contain cluster members.

Disabling automatic starting of applications

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

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

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 Mapping settings page for your application. Click **Applications > Enterprise Applications >***application_name* **> Target Mappings >***target_name*. The *target_name* is the server on which the application resides. You use the Target Mapping settings page to map an installed application or module to a server or cluster.
2. Clear the **Enabled** check box.
3. Click **OK**.
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, select the **Enabled** check box on the Target Mappings settings page for the application, click **OK**, and then save changes to the configuration.

Target mapping collection

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

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

Target

States the name of the target server or cluster to which the application or module maps. You specify the target on the Map modules to servers 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, Web module or application client module developed for a WebSphere Application Server 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 run-time 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 run-time 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.

Status
Indicates whether the status of the target server or cluster is started, stopped or unavailable.

Target mapping settings
Use this page to map a deployed application or module to a server or cluster.

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

Target:

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

Data type String

Enabled:

Indicates whether the application modules installed on the target server are started (or enabled) when the server starts. This sets the initial state of application modules. A `true` value indicates that the corresponding modules are enabled and thus are accessible when the server starts. A `false` value indicates that the corresponding modules are not enabled and thus are not accessible when the server starts.

Data type Boolean
Default true

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.

Refer to “Ways to update application files” on page 76 and decide how to update your application files. You can update enterprise applications or modules using the administrative console or a wsadmin tool. Both ways provide identical updating capabilities. Further, in some situations, you can update applications or modules without restarting the application server.

Note that Version 6 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. The administrative console Server collection″ pages show the versions for servers and cluster members. You can deploy J2EE 1.4 modules to Version 6.x servers or to clusters that contain Version 6.x cluster members only. You cannot deploy J2EE 1.4 modules to servers on Version 5.x nodes or to clusters that contain Version 5.x cluster members. Refer to “Installable module versions” on page 39 for details.

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

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. 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.
2. Back up the installed application.
   a. Go to the ″Enterprise applications collection″ 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.

3. With the application selected on the Enterprise Applications page, click **Update**. The Preparing for application update page is displayed.
4. Under **Specify the EAR, WAR or JAR module to upload and install**:
   a. Ensure that **Application name** refers to the application to be updated.
   b. Under **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. Briefly, the options are as follows:

      Full application

      Replaces the installed (old) application with the updated (new) application on the server. If you select **Full application**, specify the path for the new `.ear` file. The path provides the location of the new `.ear` file before installation.

      Single module

      Adds a new module to, or replaces a module in, the installed application. Specify the path for the new Web module (`.war`), EJB module (`.jar`), or resource adapter module (`.rar`). The path provides the location of the new module before installation.

      To replace a module, the value for **Relative path to module** (or 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 value for **Relative path to module** 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 module, specify a value for **Context root**.

      Single file

      Adds a new file to, or replaces a file in, the installed application. Specify the path for the new file. The path provides the location of the new file before installation.

      To replace a file, the value for **Relative path to file** must match the path of the file to be updated in the installed application.

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

      The relative path to a file from the root of the application is the concatenation of the module path and the file path within the module. For example, if the file is `com/mycompany/abc.class` within the module `foo.jar`, then the relative file path is `foo.jar/com/mycompany/abc.class`.

      Partial application

      Updates 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.

      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.

      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.

      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. Refer to Preparing for application update settings for more information.

After you select an option, specify a path. 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. You can browse the entire file system of a node if the node agent or deployment manager is running on that selected node. Only .ear, .jar, or .war files are shown during the browsing.

During application updating, 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, 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 on a Windows machine might be C:\WebSphere\AppServer\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 orserver machine.

5. If you selected the **Full application** or **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.
6. Click **Finish**.
7. If you did not use the Map modules to servers page of the update wizard, after updating the application, map the installed application or module to servers or clusters. Use the Map modules to servers page accessed from the Enterprise Applications page.
   a. Go to the Map modules to servers page. Click **Applications > Enterprise Applications >***application_name* **> Map modules to servers**.
   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 or to clusters that contain cluster members on Version 6.x nodes only.

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

1. Save the changes to your configuration.

   In the Network Deployment product, after you click **Save** the old application files are deleted and new files are copied when the configuration on the deployment manager synchronizes with the configuration on the node where the application is installed.

If the application is running when you update it, the application stops running before its files are copied to the destination directory of the node and restarts after the copy operation completes. Thus, the application is unavailable on the node during the time the node is synchronizing its configuration with the deployment manager.

2. Examine the values specified for **Reload Enabled** and **Reload Interval** on the settings page for your enterprise application.

   If reloading of application files is enabled and the reload interval is greater than zero (0), the application's files are reloaded after the application is updated. For Web modules such as servlets and JavaServer page (JSP) files, a Web container reloads a Web module only when the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file is also set to `true`. You can set reloadingEnabled to `true` when editing your Web module's extended deployment descriptors in an assembly tool.

3. 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.

4. If the application you are updating is deployed on a server that has its application class loader policy set to `Single`, restart the server. See ″Application server settings″ in the information center.

5. If a changed application or module is deployed on a cluster, click **Rollout Update** on the Enterprise Applications page to propagate the changed configuration on all cluster members of the cluster on which the application or module is deployed. **Rollout Update** sequentially updates the configuration on the nodes that contain cluster members.

Ways to update application files

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

Table 4. Ways to update application files

Option	Method	Comments	Starting after update
Administrative console update wizard See "Updating applications" on page 73.	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`, enterprise bean `.jar`, or connector `.rar` file. • Use **Single file** to update a file other than an `.ear`, `.war`, 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**.
wsadmin scripts	Invoke AdminApp object *install* commands with the *-update* option in a script or at a command prompt.	Getting started with scripting provides an overview of wsadmin.	Invoke the wsadmin *startApplication* command.

Table 4. Ways to update application files (continued)

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, update 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	Start the application by either of the following methods: • On the Enterprise Applications page, select the updated application and click **Start**. • Invoke the wsadmin *startApplication* command.
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.	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`, 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.

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 name

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

Full application

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

After selecting this option, 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.You can browse the entire file system of a node if the node agent or deployment manager is running on that selected node. Only `.ear`, `.jar`, 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 on a Windows machine might be `C:\WebSphere\AppServer\installableApps\test.ear`.

Note: 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`, 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.

After specifying the required information on the `.ear` file, 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.

Single module

Under **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), 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 **Full application** above.

To replace a module, the value for **Relative path to module** (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 value for **Relative path to module** 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 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`.

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.

Single file

Under **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`, `.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 **Full application** option. To update a `.war` file, `.rar` file , or `.jar` file that is defined as a module in the application, use the **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 **Full application** above.

Next, specify a value for **Relative path to file**. The relative path of the file must start 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 value for **Relative path to file** must match the path of the file to be updated in the installed application.

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

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.

Partial application

Under **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, .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 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 article assumes that your application files are deployed on a server and you want to upgrade the files.

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 73, 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.

Important: Do not use hot deployment to update components in a production deployment manager managed cell. Hot deployment is well-suited for development and testing, but poses unacceptable risks to production environments. Full or partial resynchronization might erase hot deployed components. Also, running the `restoreconfig` command might overwrite changes made to expanded application files. Further, hot deployed components are not migrated between versions of WebSphere Application Server. To add new components or modules to an enterprise application, reassemble the application EAR file so it has the new components or modules and then redeploy the EAR file.

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, *install_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. **Required:** If you are running WebSphere Application Server on a group of machines using Network Deployment and you are changing an application on a particular node, disable automatic synchronization.

a. Click **System administration > Node agents >***node_agent_name* **> File synchronization service** in the console navigation tree.
b. On the ″File synchronization service settings″ page, clear the check box for **Automatic synchronization** and click **OK**.

When you run WebSphere Application Server on a group of machines using Network Deployment and you change a file on the disk in the expanded application directory for a particular node, you can lose those changes the next time node synchronization occurs. In the Network Deployment environment, the configuration stored by the deployment manager is the master copy and any changes detected between that master copy and the copy on a particular machine trigger the master copy to be downloaded to the node.

4. **Optional:** Examine the values specified for **Enable class reloading** and **Reloading interval** on the settings page for your enterprise application. If reloading of application files is enabled and the reload interval is greater than zero (0), the application files are reloaded after the application is updated. For Web modules such as servlets and JavaServer page (JSP) files, a Web container reloads a Web module only when the IBM extension reloadingEnabled in the `ibm-web-ext.xmi` file is also set to `true`. You can set reloadingEnabled to `true` when editing your Web module's extended deployment descriptors in an assembly tool.
5. Change or add the following components or modules as needed:
   - Application files
   - WAR files
   - EJB Jar files
   - HTTP plug-in configuration files
6. For changes to take effect, you might need to start, stop, or restart an application. "Starting and stopping applications" on page 70 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.
7. If you disabled automatic synchronization in step 3, enable automatic synchronization again:
   a. Return to the File synchronization service page.
   b. Select **Automatic synchronization**.
   c. Click **OK**.

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. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server. This article 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. See ″Uninstalling applications with the wsadmin tool″ in the information center.

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. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server. This article 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. You can use the update wizard of the administrative console to make the changes without having to stop and restart the server. This article 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. 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.

Application binaries are deleted when configuration changes on the deployment manager synchronize with configurations for individual nodes.

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 from an application page is displayed
  4. Select the URI of the file that you want removed from the application.
  5. Select **Export before removing file** to back up the application.
  6. Specify the location to which you want the file exported.
  7. Click **Back** to return to the Enterprise Applications page.
- Remove a file from a module.
  1. Go to the settings page for the application. Click **Applications > Enterprise Applications >***application_name* in the console navigation tree.
  2. Under **Related Items**, click Web modules, EJB Modules, or Connector Modules.
  3. Select the module from which you want to delete a file.
  4. Click **Remove File**. The Remove a file from a module page is displayed.
  5. Select the URI of the file that you want removed from the module.

6. Optional: Back up the application. Select the application name and then specify the location to which you want the file exported.
7. 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. Application binaries are deleted when configuration changes on the deployment manager synchronize with configurations for individual nodes.

If the application or module is deployed on a cluster and you have no more configuration changes to make, click **Rollout Update** on the Enterprise Applications page to propagate the changed configuration on all cluster members of the cluster on which the application or module is deployed. **Rollout Update** sequentially updates the configuration on the nodes that contain cluster members.

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
- Programming instructions and examples

Programming model and decisions
- The J2EE™ Tutorial: The Duke's Bookstore Application
- Best Practices in WebSphere Application: Separating the developers from the administrators
- Designing Enterprise Applications with the Java™ 2 Platform, Enterprise Edition, Second Edition
- Designing Enterprise Applications, Second Edition
- Building Java™ Enterprise Applications Volume I: Architecture

Programming instructions and examples
- WebSphere Application Server education
- Developing and Testing a Complete 'Hello World' J2EE Application with IBM WebSphere Studio Application Developer for Linux
- Writing Enterprise Applications with Java™ 2 Platform, Enterprise Edition

Administration
- Listing of all IBM WebSphere Application Server Redbooks

Chapter 6. Learn about WebSphere applications

Use this section as a starting point to investigate the technologies used in and by applications that you deploy on the application server.

See ″Learn about WebSphere applications: Overview and new features″ in the information center for an introduction to each technology.

Web applications	How do I?...	Overview		Samples
EJB applications	How do I?...	Overview	Tutorials	Samples
Client applications	How do I?...	Overview		Samples
Web services	How do I?...	Overview	Tutorials	Samples
Data access resources	How do I?...	Overview	Tutorials	Samples
Messaging resources	How do I?...	Overview	Tutorials	Samples
Mail, URLs, and other J2EE resources	How do I?...	Overview		
Security	How do I?...	Overview	Tutorials	Samples
Naming and directory	How do I?...	Overview		
Object Request Broker	How do I?...	Overview		
Transactions	How do I?...	Overview		Samples
ActivitySessions	How do I?...	Overview		Samples
Application profiling	How do I?...	Overview		Samples
Asynchronous beans	How do I?...	Overview		Samples
Dynamic caching	How do I?...	Overview		
Dynamic query	How do I?...	Overview		Samples
Internationalization	How do I?...	Overview		Samples
Object pools	How do I?...	Overview		
Scheduler	How do I?...	Overview		Samples
Startup beans	How do I?...	Overview		
Work areas	How do I?...	Overview		

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.
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 ″Developing servlets with WebSphere Application Server extensions″ and the ″Resources for learning″ article for links to design documentation.
3. Develop the Web application, using WebSphere Application Server extensions to enhance its functionality. See ″Developing Web applications″ in the information center.

4. Assemble the Web application into a Web module using an assembly tool. See ″Assembling Web applications″ in the information center. 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. See ″Web container troubleshooting tips″ in the information center.
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. See ″Accessing the Samples (Samples Gallery)″ in the information center. This application, comprised of a servlet and Web pages, greets new users when the application is accessed.

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.

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

The IBM 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 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.

The URL for accessing Samples is: `http://localhost:9080/WSsamples/`

Snoop

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 execute the servlet.

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

HelloHTML

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`.

HitCount

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 6.0 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 engine:

The WebSphere Application Server JavaServer Pages (JSP) engine is the implementation of the JavaServer Pages Specification.

WebSphere Application Server 6.0 supports the JSP 2.0 specification.

The JSP engine

- Validates JSP source, both classic and XML styles
- Translates JSP source to Java classes
- Compiles Java classes, reporting any errors
- Generates Java classes for any tag files that are used by the JSP
- Interfaces with the Web container to load JSP class files
- Supports JSP batch compilation, JSP compilation during application installation, and JSP compilation during the build process of customer applications, through an Ant task.
- Loads class files, and manage life-cycle (reloading, unloading as necessary)
- Supports debugging of JavaServer Pages files through support for JSR 45 (Debugging Support for Other Languages)

JSP engine configuration parameters: In WebSphere Application Server, you can configure the JavaServer Pages (JSP) engine for optimal performance in a production server environment and for the needs of developers in a development environment. The configuration parameters are described below.

The JSP engine parameters are case sensitive. If the value specified for a parameter is comprised of two or more words separated by spaces, you must add quotation marks around the value. Some parameters affect the Java source that is generated for a JSP or tag file. These parameters are identified by the statement ″This parameter requires regeneration of Java source.″ This statement indicates that if the configuration parameter is modified, the new value for the parameter does not have any effect until the JSP files are retranslated and the Java sources are recompiled.

- **compileWithAssert**

  Specifies whether the generated Java classes should contain support for the Developer Kit, Java Technology Edition 1.4 Assertion facility. The effect of setting this parameter to true is that the –source 1.4 option is passed to the Java compiler. The default for this parameter is `false`. This parameter requires regeneration of Java source.
- **classdebuginfo**

  Indicates whether the compiler includes debugging information in the generated class file. When you set this parameter to true, the –g option is passed to the Java compiler. The default for this parameter is `false`. This parameter requires regeneration of Java source.
- **deprecation**

  Specifies whether the compiler generates deprecation warnings when compiling the generated Java source. When you set this parameter to true, the -deprecation option is passed to the Java compiler. The default for this parameter is `false`. This parameter requires regeneration of Java source.
- **disableJspRuntimeCompilation**

  If this option is set to true, the JSP engine at runtime 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 to load class files. When this option is set to true, you can install an application without JSP source, but the application must have precompiled class files. There is a Web container custom property with the same name that is 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. The default for this parameter is `false`.
- **extendedDocumentRoot**

  To allow a JSP file resource to be shared across Web application archives, specify a comma delimited list of directories and/or Java Archive (JAR) files as search paths to be used if the requested resource

cannot be located in the Web application archive's public document tree. If the JSP file is located inside a JAR file and reloadEnabled is true, the timestamp of the JAR file is used for isOutDated checks for recompile purposes. The default for this parameter is null.

- **ieClassID**

  Indicates the Java plug-in COM class ID for Internet Explorer. The `<jsp:plugin>` tags use this value. The default classid is `clsid:8AD9C840-044E-11D1-B3E9-00805F499D93`. This parameter requires regeneration of Java source.
- **javaEncoding**

  Specifies the encoding that is used when the `.java` file is generated, and when it is compiled by the Java compiler. Set this parameter when the page encoding of your JSP pages is not UTF-8 compatible. When javaEncoding is set, the encoding is passed to the Java compiler through the -encoding argument. Note that encoding is not supported by Jikes. The default is UTF-8. This parameter requires regeneration of Java source.
- **jspCompileClasspath**

  This parameter tells the JSP engine to use a small class path for the Java compilation phase. The small class path speeds up the compilation process. This small class path is not used by default because it includes only a subset of WebSphere JAR files and excludes many WebSphere JAR files that contain WebSphere public APIs.

  If your JSP files do not use WebSphere public APIs within scriptlets, you can enable the small class path by using the jspCompileClasspath parameter with no value. See the example in ″Configuring JSP engine parameters″ in the information center. If your JSP files do use WebSphere public APIs within scriptlets, then add those additional JAR files to the jspCompileClasspath option. The JAR file entries are separated by spaces, and are assumed to be relative to the WebSphere Application Server installation root.

  The entire WebSphere class path is used by default. This parameter requires regeneration of Java source.
- **jsp.file.extensions**

  For JSP files with extensions other than the four standard extensions, *.jsp, *.jspx, *.jsw, and *.jsv, you can configure this the extensions using this parameter. These extensions are added to the standard extensions. The preferred method for doing this is to create a `<jsp-property-group>`in web.xml, and add a `<url-pattern>` tag for each extension.

  The JSP engine can handle a list of file extensions that is separated by a colon or semi-colon. For example, *.ext1;*.ext2:*.extn
- **keepgenerated**

  Indicates that the Java files generated by the JSP compiler during the translation phase of the processing are retained. The default for this parameter is `false`. This parameter requires regeneration of Java source.
- **keepGeneratedclassfiles**

  Indicates that the class files generated by the JSP compiler during the translation phase of the processing are retained. The default for this parameter is `true`. This parameter requires regeneration of Java source.
- **reloadEnabled**

  Determines whether or not a JSP file is translated and compiled at runtime if the JSP file or its dependencies (see trackDependencies) are modified. If reloadEnabled is false, a JSP file is still compiled, if necessary, on the first request to it unless the parameter disableJspRuntimeCompilation is true. The default for this parameter is `false`.

  If this JSP engine parameter is not specified, the equivalent Web container parameter for Web module class reloading is used. However, for an application whose deployment descriptor is at the Servlet 2.2 level, the default is true. This is done for the support of applications being migrated from WebSphere Application Server Version 4.x.
- **reloadInterval**

  If reloading is enabled, reloadInterval 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 5 seconds prior to the current request for the JSP file. The larger the reloadInterval, the less frequently the JSP engine checks for the need to reload a JSP file. If this JSP engine parameter is not specified, the equivalent Web container parameter for Web module class reloading is used. However, for an application whose deployment descriptor is at the Servlet 2.2 level, the default is 5 seconds. This is done for the support of applications being migrated from WebSphere Application Server Version 4.x.

- **scratchdir**

  Specifies the directory where the generated class files are created. The system property com.ibm.websphere.servlet.temp.dir is used to set the scratchdir option on a server-wide basis. The JSP engine scratchdir parameter takes precedence over the system property com.ibm.websphere.servlet.temp.dir. The default for this parameter is `{WAS_ROOT}/profiles/profilename/temp`. This parameter requires regeneration of Java source.
- **trackDependencies**

  If reloading is enabled, trackDependencies determines whether the JSP engine tracks modifications to the requested JavaServer Pages files dependencies as well as to the JSP file itself. The dependencies tracked by the JSP engine are :
  1. files statically included in the JSP file
  2. tag files referenced in the JSP file (excluding tag files that are in JARs)
  3. TLD files referenced in the JSP file (excluding TLDs that are in JARs)

  The default is `false`.
- **useFullPackageNames**

  If useFullPackageNames is true, the JSP engine generates and loads JSP classes using full package names. The default is to generate all JSP classes in the same package. (For more information, see “Packages and directories for generated .java and .class files” on page 102). The JSP engine’s class loader knows how to load JSP classes when they are all in the same package.

  The default method of generating all JSP classes in the same package has the benefit of generating smaller file-system paths. Full package names has the benefit of enabling the configuration of precompiled JSP class files as servlets in the `web.xml` file without the use of the jsp-file attribute, resulting in a single class loader, the Web application’s class loader, that is 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 files 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 file that was successfully translated and compiled. The information can optionally be copied into the Web module’s `web.xml` file so that the JSP files are loaded as servlets by the Web container. Note that if a JSP file is configured as a servlet in this way, no reloading of the JSP file is done at runtime if the JSP file is modified. This is because the JSP file is treated as a regular servlet and requests for it do not pass through the JSP engine. This parameter requires regeneration of Java source.
- **useImplicitTagLibs**

  The JSP engine implicitly recognizes tsx and jsx as tag library prefixes for tag libraries supplied by the JSP engine. If tsx or jsx are used as prefixes for a customer‘s tag library, the customer‘s tag library overrides the implicit tag library. However, the implicit tag library is still cached by the JSP engine. Explicitly setting this parameter to false tells the engine not to cache the implicit tag library, and save resources. The default for this parameter is `true`.
- **useJikes**

  Specifies whether Jikes is used for compiling Java sources. NOTE: Jikes is not shipped with WebSphere Application Server. The default for this parameter is `false`. This parameter requires regeneration of Java source.
- **usePageTagPool**

  *Enables or disables the reuse of custom tag handlers on an individual JavaServer Pages basis. The default for this parameter is `false`. This parameter requires regeneration of Java source.
- **useThreadTagPool**

*When thread-level tag handler pooling is used, tag handlers may be reused among separate occurrences of a custom action across all JSP pages in a single Web module across separate requests. The default for this parameter is `false`. This parameter requires regeneration of Java source.

- **verbose**

  Indicates that the compiler generates verbose output when compiling the generated Java source code. The effect of setting this parameter to true is that the -verbose option is passed to the Java compiler. The default for this parameter is `false`. This parameter requires regeneration of Java source.

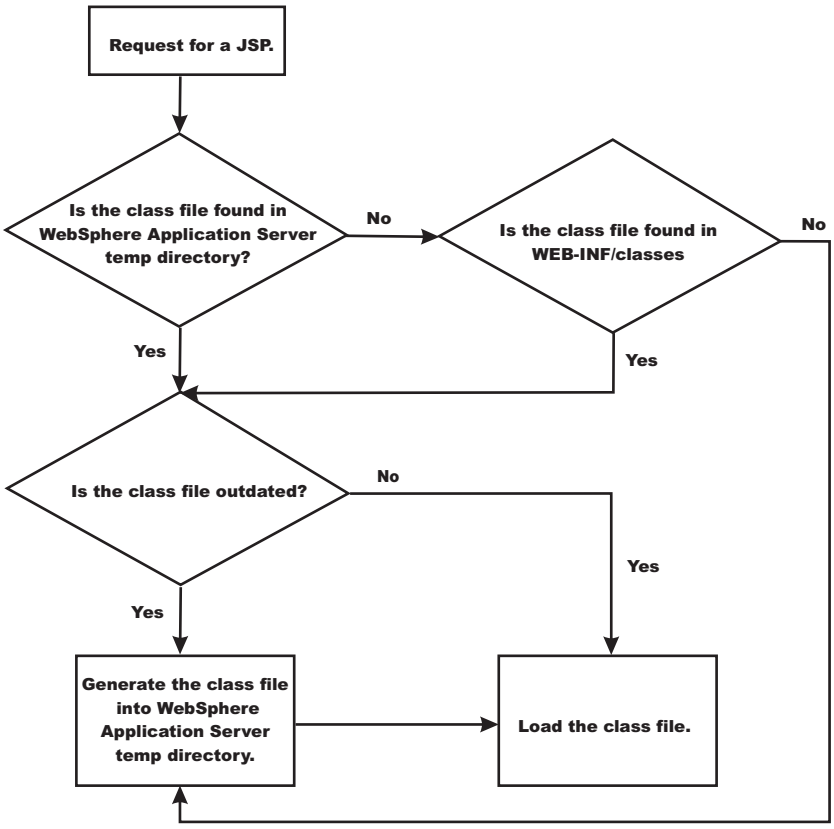
*Enabling custom tag handler reuse might reveal problems in the tag handler code with regard to the tag's ability to be reused. A custom tag handler should always do two things:

- The release method of the tag handler should reset its state and release any private resources that it might have used. The JSP engine ensures the release method is called before the tag handler is garbage collected.
- In the doEndTag method, all instance states associated with this instance must be reset.

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.

You should not use the `CANCEL appserver_proc_name` command to stop a server. Every time a server is cleanly stopped, these `temp` directories are removed. However, if the server is frequently not stopped cleanly, which happens if you cancel rather than stop the server, these directories are not removed and the HFS used for the temp directory eventually becomes full.

You can also prevent this storage problem from occurring if you precompile your JSPs when you install an application or if you use the JspBatchCompiler function to precompile them before they are invoked.



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 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 104 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**	**useFullPackage Names=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.

```
/*
C:/WebSphere_6.0/AppServer/profiles/AppSrv01/installedApps/MyCell/sampleApp.ear/examples.war/
WEB-INF/classes/_ibmjsp/_jsp1.java was generated @ Thu Oct 14 10:05:56 EDT 2004
IBM WebSphere Application Server - ND, 6.0.0.0
    Build Number: o0441.04
    Build Date: 10/12/04

******************************************************
The JSP engine configuration parameters were set as follows:

classDebugInfo =              [false]
debugEnabled =                [false]
deprecation =                 [false]
compileWithAssert =           [false]
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]
jspCompileClasspath =         []
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?	useFull PackageNames
<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. In WebSphere Application Server 6.0, 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.

1. An example of a configuration repository directory is
   {WAS_ROOT}/profiles/profilename/config/cells/cellname/applications/
   enterpriseappname/deployments/deployedname/webmodulename
2. 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 runtime reloading: JSP files can be translated and compiled at runtime 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 JSPs 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 runtime 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 runtime or is processed by the batch compiler.

Disabling compilation

Disablement of runtime 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 runtime 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.

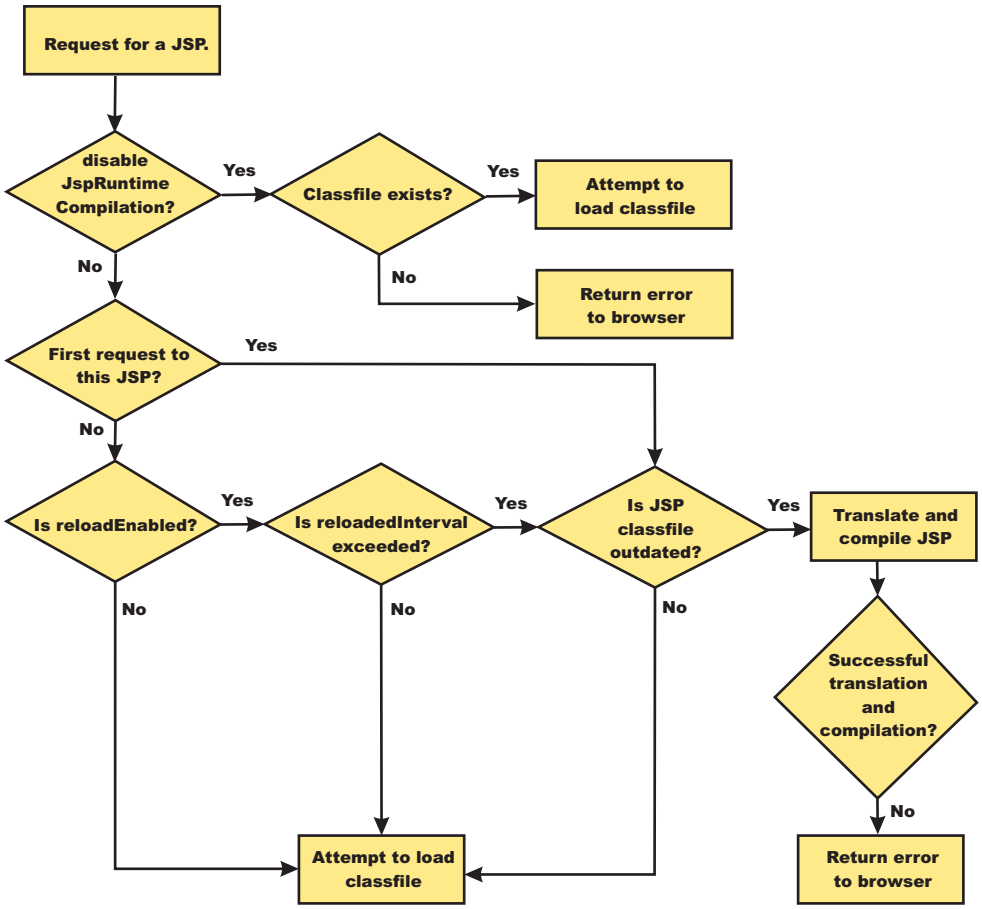


Figure 1. Reload processing sequence when **trackDependencies** *is false.*

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.

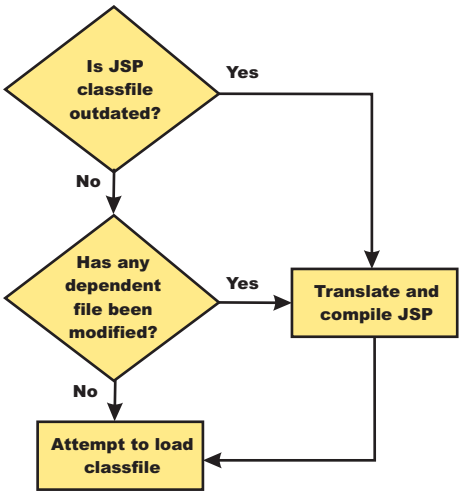


Figure 2. Additional reload processing performed when **trackDependencies** *is true.*

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 joblog (sysprint) file if ras_trace_outputLocation in `was.env` file is set to SYSPRINT or to CTRACE if ras_trace_outputLocation is set to BUFFER.

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.

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.

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 (described in the topic Batch Compiler Ant Task) is also available for executing the batch compiler using Ant.

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>]
[-jspCompileClasspath <classpath to Websphere Application Server public API JAR files; or no value at all>]
[-verbose <true | false>]
[-deprecation <true | false>]
[-javaEncoding <encoding>
[-compileWithAssert <true | false>]
[-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” on page 98.
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 runtime, 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:

where:

- **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. A template response file, batchcompiler.properties.default, is found in {WAS_ROOT}/bin. 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.

  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 will be 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.[bat/sh]` in `{WAS_ROOT}/bin`. 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 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 runtime 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 runtime 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 runtime, these same JAR files and classes have to be made available through the standard WebSphere Application Server configuration mechanisms.

Default: null

- **jspCompileClasspath**

  This option instructs the batch compiler to use a small class path for the Java compilation phase. The small class path greatly speeds up the compilation process. This small class path is not used by default because it includes only a subset of WebSphere Application Server JAR files. The small class path excludes many WebSphere Application Server JAR files, among which are those that contain WebSphere public APIs.

  If your JSP pages do not use any WebSphere public APIs within scriptlets you can enable the small class path by using the jspCompileClasspath parameter with no value, as in Example 1 below.

  If your JSP pages do use WebSphere public APIs within scriptlets, then add those additional JAR files to the jspCompileClasspath option, as in Example 2 below.

  The entries are separated by spaces, and are assumed to be relative to the WebSphere Application Server installation root.

  Example 1: If no public APIs are needed in JSP pages: JspBatchCompiler -enterpriseApp.name sampleApp -jspCompileClasspath

  Example 2: public APIs from the `admin.jar` file needed in JSP pages: JspBatchCompiler -enterpriseApp.name sampleApp –jspCompileClasspath ″lib/admin.jar″

  Use-case: Use this parameter to speed up the Java compilation step of JSP page processing.

  Default: The entire WebSphere Application Server class path is used by default.

- **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.

- **compileWithAssert**

  Tells the batch compiler to enable assertions. If compileWithAssert is true, the batch compiler will pass the –source 1.4 option to the javac compiler. If compileWithAssert is false, no option is sent to the javac compiler. The default behavior of javac is to compile code normally even if the word assert is used as regular identifier.

  Example: JspBatchCompiler -war.path c:\myproject\examples.war -compileWithAssert true

  Use-case: Set this parameter to true when you want you use the assertion facility in your JSP pages and you want to be able to turn on assertions at runtime.

  Default value: false

- **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 the section, “JSP batch compiler tool” on page 110, 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
jspCompileClasspath	-jspCompileClasspath
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

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:

```
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"
            jspCompileClasspath=""

  />
 </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"
            jspCompileClasspath=""

  />
 </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"
            jspCompileClasspath=""

  />
 </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
jspCompileClassPath parameter	When this parameter is used, the only change to the information above is that a subset of WebSphere Application Server JAR files and classes is used for Java compilation. All JAR files and classes, that are given in the value for the *jspCompileClassPath* parameter are also added to the class path for Java compilation.		

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: 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.

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: Use the <tsx:dbquery> tag to establish a connection to a database, submit database queries, and return the results set.

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: The <tsx:dbmodify> tag establishes a connection to a database and then adds records to a database table.

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: The `<tsx:getProperty>` tag gets the value of a bean to display in a JavaServer Pages (JSP) file.

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: 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.

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: The <tsx:getProperty> tag repeats a block of HTML tagging.

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: 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: 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: 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>
```

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 or 6.

Symptom Your Web module does not run when you migrate it to Version 5 or 6

Problem In Version 4.x, the classpath setting that affected visibility was *Module Visibility Mode*. In Versions 5 and 6, 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 the articles “Class loaders” on page 25 and Chapter 4, “Class loading,” on page 25 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 article Assembling Web applications for more information.

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
- Redbook on Servlet and JSP file Programming
- 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

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, session recovery, and session clustering.
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.

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.

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.
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.

Distributed sessions

WebSphere Application Server provides the following session mechanisms in a distributed environment:

- **Database Session persistence**, where sessions are stored in the database specified.
- **Memory-to-memory Session replication**, where sessions are stored in one or more specified WebSphere Application Server instances.

When a session contains attributes that implement HttpSessionActivationListener, notification occurs anytime the session is activated (that is, session is read to the memory cache) or passivated (that is, session leaves the memory cache). Passivation can occur because of a server shutdown or when the session memory cache is full and an older session is removed from the memory cache to make room for a newer session. It is not guaranteed that a session is passivated in one application server prior to being activated in another.

Session recovery support

For session recovery support, WebSphere Application Server provides distributed session support in the form of database sessions and memory-to-memory replication. 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.

Distributed environment settings:

Use this page to specify a type for saving a session in a distributed environment.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Web container settings** > **Session management** > **Distributed environment settings**.

Distributed sessions:

Specifies the type of distributed environment to be used for saving sessions.

None	Specifies that the session management facility discards the session data when the server shuts down.
Database	Specifies that the session management facility stores session information in the data source specified by the data source connection settings. Click **Database** to change these data source settings.
Memory-to-memory replication	Specifies that the session management facility stores the session information in a data source in memory. The session information is copied to other session management facilities for failure recovery.

Memory-to-memory replication

WebSphere Application Server supports session replication to another WebSphere Application Server instance. This support is referred to as *memory-to-memory session replication*. In this mode, sessions can replicate to one or more WebSphere Application Server instances to address HTTP Session single point of failure (SPOF).

The WebSphere Application Server instance in which the session is currently processed is referred to as the *owner of the session*. In a clustered environment, session affinity in the WebSphere Application Server plug-in routes the requests for a given session to the same server. If the current owner server instance of the session fails, then the WebSphere Application Server plug-in routes the requests to another appropriate server in the cluster. In a peer-to-peer cluster, the hot failover feature causes the plug-in to failover to a server that already contains the backup copy of the session, avoiding the overhead of session retrieval from another server containing the backup. In a client/server cluster, the server retrieves the session from a server that has the backup copy of the session. The server now becomes the owner of the session and affinity is now maintained to this server.

There are three possible modes. You can set up a WebSphere Application Server instance to run in:
- **Server mode:** Only store backup copies of other WebSphere Application Server sessions and not to send out copies of any session created in that particular server
- **Client mode:** Only broadcast or send out copies of the sessions it owns and not to receive backup copies of sessions from other servers
- **Both mode:** Simultaneously broadcast or send out copies of the sessions it owns and act as a backup table for sessions owned by other WebSphere Application Server instances

You can select the replication mode of server, client, or both when configuring the session management facility for memory-to-memory replication. The default is both. This storage option is controlled by the mode parameter.

The memory-to-memory replication function is accomplished by the creation of a data1 replication service instance in an application server that talks to other data replication service instances in remote application servers. You must configure this data replication service instance as a part of a replication domain. See ″Replication domain collection″ in the information center. Data replication service instances on disparate application servers that replicate to one another must be configured as a part of the same domain. You must configure all session managers connected to a replication domain to have the same topology. If one session manager instance in a domain is configured to use the client/server topology, then the rest of the session manager instances in that domain must be a combination of servers configured as Client only and Server only. If one session manager instance is configured to use the peer-to-peer topology, then all session manager instances must be configured as Both client and server. For example, a server only data

replication service instance and a both client and server data replication service instance cannot exist in the same replication domain. Multiple data replication service instances that exist on the same application server due to session manager memory-to-memory configuration at various levels that are configured to be part of the same domain must have the same mode.

With respect to mode, the following are the primary examples of memory-to-memory replication configuration:

- Peer-to-peer replication
- Client/server replication

Although the administrative console allows flexibility and additional possibilities for memory-to-memory replication configuration, only the configurations provided above are officially supported.

There is a single replica in a cluster by default. You can modify the number of replicas through the replication domain.

HTTP session replication in the controller

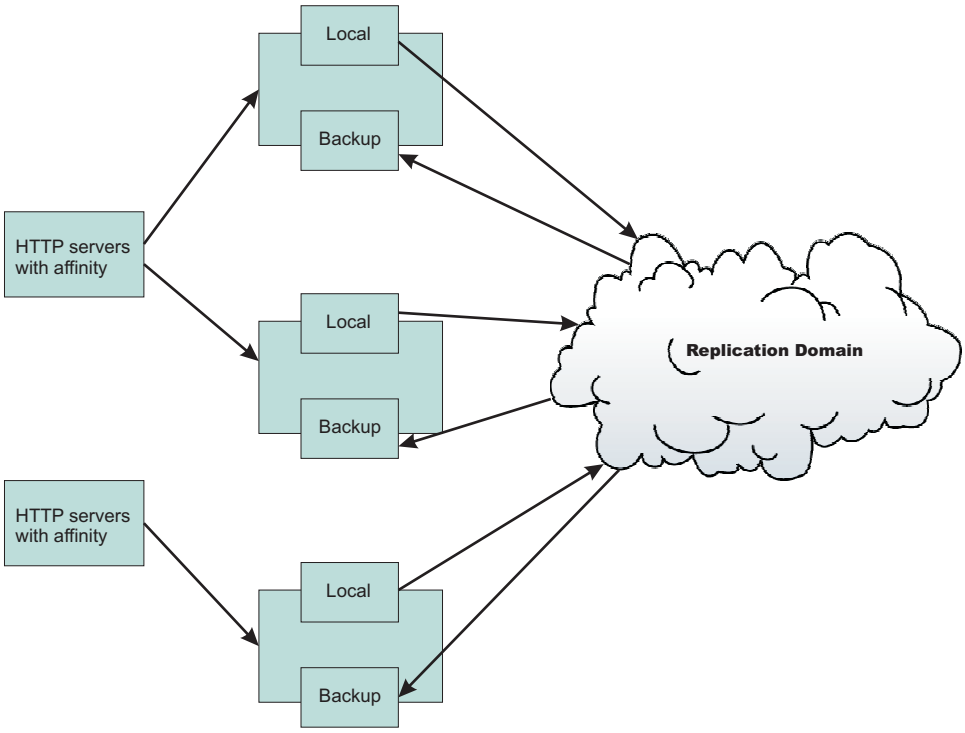
WebSphere Application Servers on z/OS that are enabled for HTTP session memory-to-memory replication can store replicated HTTP session data in the controller and replicate data to other WebSphere Application Servers. HTTP session data that is stored in a controller is retrievable by any of the servants of that controller. HTTP session affinity is still associated to a particular servant; however, if that servant should fail, any of the other servants can retrieve the HTTP session data stored in the controller and establish a new affinity.

The capability of storing HTTP sessions in the controller can also be enabled in unmanaged application servers on z/OS. When this capability is enabled, servants store the HTTP session data in the controller for retrieval when a servant fails which is similar to managed servers. HTTP session data stored in the controller of an unmanaged application server is not retrievable by other application servers and is not replicated to other application servers.

The capability to store HTTP session data in the controller in an unmanaged application server is enabled by setting the JVM custom property HttpSessionEnableUnmanagedServerReplication to true. You can set this property at **Servers > Application servers >***server_name* **> Java and Process Management > Process Definition > Servant > Java Virtual Machine > Custom Properties**.

Memory-to-memory topology: Peer-to-peer function:

The basic peer-to-peer (both client and server function, or both mode) topology is the default configuration and has a single replica. However, you can also add additional replicas by configuring the replication domain.



In this basic peer-to-peer topology, each server Java Virtual Machine (JVM) can:

- Host the Web application leveraging the HTTP session
- Send out changes to the HTTP session that it owns
- Receive backup copies of the HTTP session from all of the other servers in the cluster

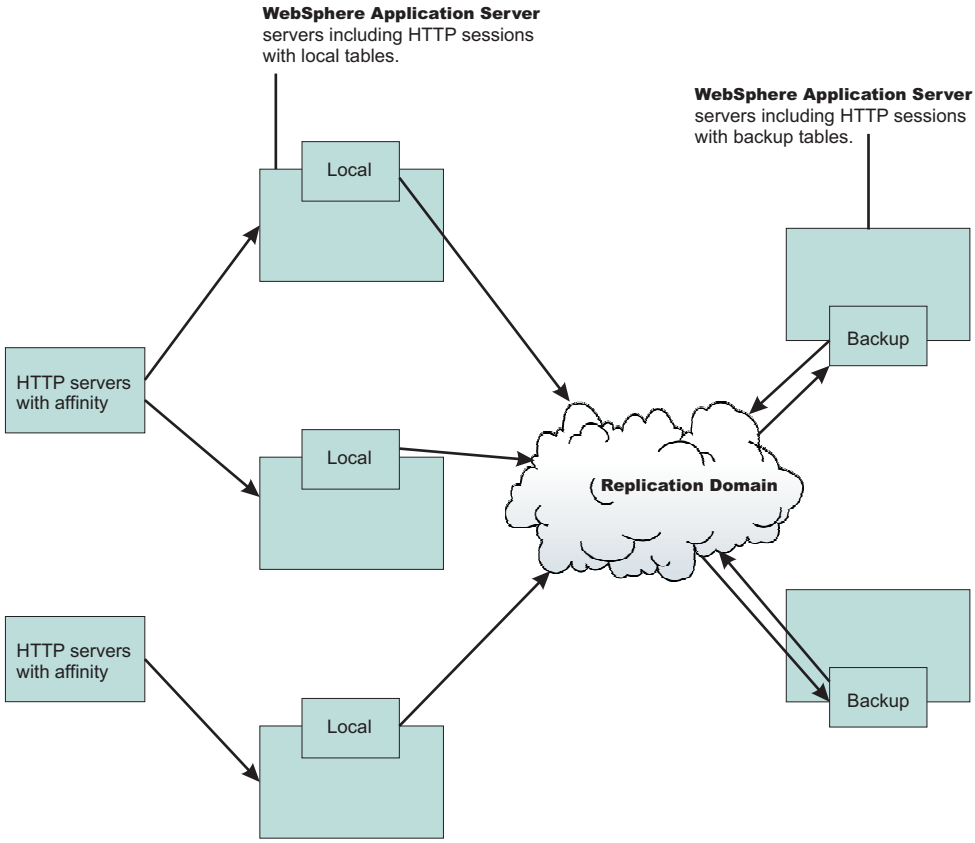
This configuration represents the most consolidated topology, where the various system parts are collocated and requires the fewest server processes. When using this configuration, the most stable implementation is achieved when each node has equal capabilities (CPU, memory, and so on), and each handles the same amount of work.

Session hot failover

A new feature called session hot failover has been added to this release. This feature is only applicable to the peer-to-peer mode. In a clustered environment, session affinity in the WebSphere Application Server plug-in routes the requests for a given session to the same server. If the current owner server instance of the session fails, then the WebSphere Application Server plug-in routes the requests to another appropriate server in the cluster. For a cluster configured to run in the peer-to-peer mode this feature causes the plug-in to failover to a server that already contains the backup copy of the session, therefore avoiding the overhead of session retrieval from another server containing the backup.

You must upgrade all WebSphere Application Server plug-in instances that front the Application Server cluster to version 6.0 to ensure session affinity when using the peer-to-peer mode.

Memory-to-memory topology: Client/server function: The following figure depicts the client/server mode. There is a tier of applications servers that host Web applications using HTTP sessions, and these sessions are replicated out as they are created and updated. There is a second tier of servers without a Web application installed, where the session manager receives updates from the replication clients.



Benefits of the client/server configuration include:

Isolation (for failure recovery)
In this case we are isolating the handling of backup data from local data; aside from isolating the moving parts in case of a catastrophic failure in one of them, you again free up memory and processing in the servers processing the Web application

Isolation for stopping and starting
You can recycle a backup server without affecting the servers running the application (when there are two or more backups, failure recovery is possible), and conversely recycle an application JVM without potentially losing that backup data for someone.

Consolidation
There is most likely no need to have a one-to-one correspondence between servers handling backups and those processing the applications; hence, you are again reducing the number of places to which you transfer the data.

Disparate hardware:
While you run your Web applications on cheaper hardware, you may have one or two more powerful computers in the back end of your enterprise that have the capacity to run a couple of session managers in replication server mode; allowing you to free up your cheaper Web application hardware to process the Web application.

Timing consideration: Start the backup application servers first to avoid unexpected timing windows. The clients attempt to replicate information and HTTP sessions to the backup servers as soon as they come up. As a result, HTTP sessions that are created prior to the time at which the servers come up might not replicate successfully.

Memory-to-memory session partitioning

Session partitioning gives the administrator the ability to filter or reduce the number of destinations that the session object gets sent to by the replication service. You can also configure session partitioning by specifying the number of replicas on the replication domain. The Single replica option is chosen by default. Since the number of replicas is global for the entire replication domain, all the session managers connected to the replication domain use the same setting.

Single replica

You can replicate a session to only one other server, creating a single replica. When this option is chosen, a session manager picks another session manager that is connected to the same replication domain to replicate the HTTP session to during session creation. All updates to the session are only replicated to that single server. This option is set at the replication domain level. When this option is set, every session manager connected to this replication domain creates a single backup copy of HTTP session state information on a backup server.

Full group replica

Each object is replicated to every application server that is configured as a consumer of the replication domain. However, this topology is the most redundant because everyone replicates to everyone and as you add servers, more overhead (both CPU and memory) is needed to deal with replication. This mode is most useful for dynamic caching replication.

Specific number of replicas

You can specify a specific number of replicas for any entry that is created in the replication domain. The number of replicas is the number of application servers that the user wants to use to replicate in the domain. This option eliminates redundancy that occurs in a full group replica and also provides additional backup than a single replica. The number of replicas cannot exceed the total number of application servers in the cluster.

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.

Tuning session management

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.
- With distributed sessions (meaning, when sessions are stored in a database or in another WebSphere Application Server instance); it also specifies the cache size and the number of last access time updates saved in manual update mode.

For distributed sessions, when the session cache has reached its maximum size and a new session is requested, the Session Management facility removes the least recently used session from the cache to make room for the new one.

General memory requirements for the hardware system, and the usage characteristics of the e-business site, determines the optimum value.

Note that increasing the base in-memory session pool size can necessitate increasing the heap sizes of the Java processes for the corresponding WebSphere Application Servers.

Overflow in non-distributed sessions

By default, the number of sessions maintained in memory is specified by base in-memory session pool size. If you do not wish to place a limit on the number of sessions maintained in memory and allow overflow, set overflow to *true*.

Allowing an unlimited amount of sessions can potentially exhaust system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your site and creates sessions, but ignores any cookies or encoded URLs and never utilizes the same session from one HTTP request to the next.

When overflow is disallowed, the Session Management facility still returns a session with the HttpServletRequest getSession(true) method when the memory limit is reached, and this is an invalid session that is not saved.

With the WebSphere Application Server extension to HttpSession, com.ibm.websphere.servlet.session.IBMSession, an isOverflow method returns *true* if the session is such an invalid session. An application can check this status and react accordingly.

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

- **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.**

  This action allows the object to properly serialize when using distributed sessions. Without this extension, the object cannot serialize correctly and throws an error. 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.
- **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.)

Managing 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
- Best practices
- HTTP Session Persistence Best Practices
- 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
- Transaction class mappings

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 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.
3. If your application uses transaction class mappings to classify workload, in the administrative console, click **Resources > Asynchronous beans > Work managers >** *workmanager_name*. Then enter the name of the transaction class mapping file in the Default transaction class field.

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.

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 Java Server Pages (JSP) files, as well as metadata about the entry, including timeout and entry priority information.

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* > **Web modules** > *Web_module_instance*.

URI:

Specifies a URI that, when resolved relative to the application URL, specifies the location of the module archive contents on a file system. The URI must match the module URI in the deployment descriptor of an application if the module was packaged as part of a deployed application or enterprise archive (EAR) file.

Alternate deployment descriptor:

Specifies the file name for an alternative deployment descriptor file to use instead of the original deployment descriptor file in the module JAR file.

This file is the *post-assembly* version of the deployment descriptor file. You can edit the original deployment descriptor file to resolve dependencies and security information. Specifying the use of the alternative deployment descriptor keeps the original deployment descriptor file intact.

The value of the *Alternate deployment descriptor* property must be the full path name of the deployment descriptor file, relative to the module root directory. By convention, the file is in the `ALT-INF` directory. If this property is not specified, the deployment descriptor file is read from the module JAR file.

Starting weight:

Specifies the order in which modules are started. Lower weighted modules are started before higher weighted modules.

Class loader mode:

Specifies whether the class loader should search in the parent class loader or in the application class loader first to load a class. The standard for JDK class loaders and WebSphere class loaders is Parent First. By specifying Parent Last, your application can override classes contained in the parent class loader, but this action can potentially result in ClassCastException or LinkageErrors if you have mixed use of overriden classes and non-overriden classes.

The options are Parent First and Parent Last. 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	Parent First

Web container advanced settings

Use this page to support Web container advanced settings. This support includes Network QoS and transaction class mapping

To view this administrative console page, click **Servers** > **Application Servers** > *server name* > **Web Container** > **Advanced Settings**.

Network QoS:

Specifies the parameter that will be used to classify outbound data that is delivered in response to HTTP and HTTPS requests.

The classification parameters are used to construct an ApplicationData parameter for the TCP/IP network service, which is called Quality of Service (QoS). The ApplicationData parameter is used in a QoS PolicyRule statement.

You can specify at most one classification parameter. If you do not specify a classification parameter, the response data will not be classified to the network agent.

Parameter	Description
HOST	Indicates that the Host value from the Host header, not including the port, is to be used to construct an ApplicationData parameter. If you specify this parameter, WebSphere Application Server for z/OS classifies the outbound response data by using the HOST value. In the request: `http://www.mycompany.com/mywebap/myservlet` `www.mycompany.com` represents the host value.

URI
Indicates that the part of the Universal Resource Locator that specifies the path to a resource is to be used to construct an ApplicationData parameter. If you specify this parameter, WebSphere Application Server for z/OS classifies the outbound response data by using the URI value. The path must be specified exactly as it is entered in a browser because the check for this path is case sensitive.

In the request:

```
http://www.mycompany.com/mywebap/myservlet
```

`/mywebap/myservlet` represents the URI value.

HOSTURI
Indicates that the HOST and URI, concatenated together, are to be used to construct an ApplicationData parameter. If you specify this parameter, WebSphere Application Server for z/OS classifies the outbound response using the concatenated HOST and URI value.

In the request:

```
Get request: http://www.mycompany.com/mywebap/myservlet
```

`www.mycompany.com/mywebap/myservlet` represents the concatenated HOST and URI value.

TCLASS
Indicates that a valid Workload Management (WLM) transaction class is to be used to construct an ApplicationData parameter. If you specify this parameter, you must specify the fully qualified name of the transaction class mapping file on the Transaction Class Mapping property.

Transaction Class Mapping:

Specifies the fully qualified name of the file that contains the rules for classifying the Workload Management Transaction Class for HTTP or HTTPS requests. The file name is class sensitive.

For example, if `tclass.conf` is the name of your transaction class mapping file, you would specify the following for the value on this property:

`/mydir/tclass.conf`

where *mydir* is the fully qualified directory where the `tclass.conf` file is located.

For example

/mydir/tclass.conf

Web container custom properties

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 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.

To view this administrative console page, click **Servers > Application servers >***server_name* **> Web container settings > Web container > Custom properties**.

HTTP Transport custom properties can also be set at the Web container level. See ″HTTP transport custom properties″ for a description of these properties.

Name:

Specifies the name or key for the property.

Value:

Specifies the value that is paired with the specified name.

Data type String

Description:

Specifies information about the name-value pair.

Data type String

Global settings for internal servlets

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 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″, 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.

Transaction class mapping file entries

Following is the syntax for entries in a transaction class mapping file:

```
TransClassMap host:port uritemplate tclass
```

where:

host Is the value compared against the hostname of the HOST: header of the request. This value can be a wildcard '*'.

Note: A value of '*' for the host:port value is acceptable and is equivalent to '*:*'.

port Is the value compared against the port of the request. This value can be a wildcard '*'.

uritemplate
Is the value compared against the URI of the request. Any query string will not be used in the comparison. This value can be a wildcard '*', or end in a wildcard.

tclass Is the Workload Manager Transaction Class name that will be used in the creation of the enclave.

Examples:

```
TransClassMap www.ibm.com:80 /webap1/myservlet TCLASS1

TransClassMap www.ibm.com:* /webap1/myservlet TCLASS2

TransClassMap *:443 * TCLASS3

TransClassMap *:* /webap1/myservlet TCLASS4

TransClassMap www.ibm.com:* /webap2/* TCLASS5

TransClassMap * /myservlet TCLASS6

TransClassMap * * TCLASS6
```

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**.
     b. Select a server from the list of application servers.
     c. Under Additional Properties, click **Web Container**.
   - For the enterprise application level:
     a. Click **Applications** > **Applications**.
     b. Select an applications from the list of applications.
   - For the Web module level:
     a. Click **Applications** > **Enterprise Applications**.
     b. Select an applications from the list of applications.
     c. Under Related Items, click **Web Modules**.
     d. 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**.
5. Define the session recovery characteristics.

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.

Restriction: 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**.

Override session management:

Specifies whether or not these session management settings take precedence over those normally inherited from a higher level for the current application or web module.

By default, web modules inherit session management settings from the application level above it, and applications inherit session management settings from the Web container level above it.

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. 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.	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 **Modify** 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	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.

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 120 seconds, or 2 minutes. 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 executed normally or aborted in the event of a timeout. If this box is checked, the servlet executes 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 HTTP 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: 40.

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 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.

NoAffinitySwitchBack

Set this property to ″true″ to maintain affinity to the new member even after original one comes back up. When a cluster member fails, its requests routed to a different cluster member, and sessions are activated in that other member. Thus, session affinity is maintained to the new member, and when failed cluster member comes back up, the requests for sessions that were created in the original cluster member are routed back to it. Allowed values, true or false. Default: false.

It is recommended that you set this property to ″true″ when distributed sessions with time-based write is configured. Note that this property has no affect on the behavior when distributed sessions is not enabled.

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**.
3. Select a server from the list of application servers.
4. Under Container Settings, click **Web Container Settings > Web container**
5. Under Additional Properties, click **Custom Properties**.
6. Add the appropriate properties from the following list:
   - HttpSessionIdLength
   - SessionRewriteIdentifier
   - HttpSessionCloneId
   - CloneSeparatorChange
   - NoAdditionalSessionInfo
   - SessionIdentifierMaxLength
7. 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 the z/OS DB2 database containing the DB2 table for session persistence, using 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.

   Example configuration for session persistence:

Name	Sessions
JNDI Name	jdbc/sessions
Container managed persistence	Selected
Component-managed Authentication Alias	CELL/jaasalias
Container-managed Authentication Alias	CELL/jaasalias

3. Verify that the correct database is listed for the value of the **databaseName** property under **Data Sources >** *datasource_name* **> Custom Properties**. If necessary, contact your database administrator to verify the correct database name.

   For example:

Database Name	LOC1 (specify your system)

4. Create a DB2 table in the z/OS DB2 database that will be used for session persistence.
5. Configure the DB2 table for session persistence.
6. Go to the appropriate level of Session Management.

7. Click **Distributed Environment Settings**
8. Select and click **Database**.
9. Specify the Data Source JNDI name from step 3.
10. Switch to a multirow schema.
11. Click **OK**.
12. If you want to change the tuning parameters, click **Custom Tuning Parameters** and select a setting or customize one.
13. Click **Apply**.
14. 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 and recreate the database table or delete all the rows in the database table that the product uses to maintain HttpSession objects.

   See the *DB2 UDB for OS/390 and z/OS V7 Administration Guide* for a description of how to drop a DB2 database table.

   Creating a DB2 table for session persistence describes how to create a new DB2 database table.

Creating a DB2 table for session persistence

If you are using DB2 for session persistence, a DB2 table, in which session data will be collected, must be created and defined to the application server.

To create a DB2 table for collecting session data, do the following:

1. Have your DB2 Administrator create a DB2 database table for storing your session data. (For more information about creating DB2 databases see the *DB2 UDB for OS/390 and z/OS V7 Administration Guide*.)

   The table space in which the database table is created must be defined with row level locking (`LOCKSIZE ROW`). It should also have a page size that is large enough for the objects that will be stored in the table during a session. Following is an example of a table space definition with row level locking specified and a buffer pool page size of 32K:

```
CREATE DATABASE database_name
  STOGROUP SYSDEFLT
  CCSID EBCDIC;

CREATE TABLESPACE tablespace_name IN database_name
  USING STOGROUP group_name
  PRIQTY 512
  SECQTY 1024
  LOCKSIZE ROW
  BUFFERPOOL BP32K;
```

   The Session Manager will use the DB2 table defined within this table space to process the session data. This table must have the following format:

```
CREATE TABLE database_name.table_name  (
  ID               VARCHAR(95) NOT NULL ,
  PROPID           VARCHAR(95) NOT NULL ,
  APPNAME          VARCHAR(64) ,
  LISTENERCNT      SMALLINT ,
  LASTACCESS       DECIMAL(19,0),
  CREATIONTIME     DECIMAL(19,0),
  MAXINACTIVETIME  INTEGER ,
```

```
USERNAME         VARCHAR(256) ,
SMALL            VARCHAR(3122)  FOR BIT DATA ,
MEDIUM           VARCHAR(28869) FOR BIT DATA ,
LARGE            BLOB(2097152),
SESSROW          ROWID NOT NULL GENERATED ALWAYS
)
IN database_name.tablespace_name;
```

Note: The length attributes specified for VARCHAR in this example are not necessarily the values your DB2 Administrator should use for the DB2 table he is creating. See the DB2 SQL Reference for the version of DB2 you will be using for guidance in determining appropriate values for these length attributes for your installation.

A unique index must be created on the `ID` and `PROPID` columns of this table. The following is an example of the index definition:

```
CREATE UNIQUE INDEX database_name.index_name.
      database_name.table_name
      (ID     ASC,
      PROPID  ASC,
      APPNAME ASC);
```

Note:

a. At run time, the Session Manager will access the target table using the identity of the J2EE server in which the owning Web application is deployed. Any Web container that is configured to use persistent sessions should be granted both read and update access to the subject database table.

b. HTTP session processing uses the index defined using the `CREATE INDEX` statement to avoid database deadlocks. In some situations, such as when a relatively small table size is defined for the database, DB2 may decide not to use this index. When the index isn't used, database deadlocks can occur. If this situation occurs, see the DB2 Administration Guide for the version of DB2 you are using for recommendations on how to calculate the space required for an index, and adjust the size of the tables you are using accordingly.

c. It may be necessary to tune DB2 in order to make efficient use of the sessions database table and to avoid deadlocks when accessing it. Your DB2 Administrator should refer to the DB2 Administration Guide for specific information about tuning the version of DB2 you are using.

A large object (LOB) table space must be defined and an auxiliary table must be defined within that table space. The following is an example of the LOB table space definition:

```
CREATE LOB TABLESPACE LOB_tablespace_name IN database_name
    BUFFERPOOL BP32K
    USING STOGROUP group_name
    PRIQTY 512
    SECQTY 1024
    LOCKSIZE LOB;

CREATE AUX TABLE database_name.aux_table_name
    IN database_name.LOB_tablespace_name
    STORES database_name.table_name
    COLUMN LARGE;
```

An index must be created for this auxiliary table. The following is an example of the index definition:

```
CREATE INDEX database_name.aux_index_name ON
      database_name.aux_table_name;
```

2. Have your DB2 Administrator grant the the z/OS userID, under which the server region is running, the appropriate access to this DB2 table. For example,issue the following command to grant z/OS userID CBASRU1, under which the server region is running, access to the table SESSIONS contained in the database SESSDB:

   ```
   GRANT ALL ON SESSDB.SESSIONS TO CBASRU1;
   ```

3. Configure DB2 table for session persistence.

Configuring a DB2 table for session persistence

If you are using DB2 for session persistence, a DB2 table, in which session data will be collected, must be created and defined to the application server, see “Creating a DB2 table for session persistence” on page 160, for more information.

To configure a DB2 table for collecting session data, use the administrative console to add the name of this DB2 table to the Web container’s configuration properties:

1. Open the administrative console.
2. Click **Servers** > **Application Servers**.
3. Select a server from the list of application servers.
4. Under Additional Properties, click **Web Container**.
5. Under Additional Properties, click **Custom Properties**.
6. Check **SessionTableName** and then click **New**.
7. In the **Value** field, enter the name of the DB2 Session Table if you are not using the default value **SESSION**. The name must be in the form *database_name.table_name*. For example, if the database name is SESSDB and the table name is SESSIONS, enter SESSDB.SESSIONS for **Value**.Optionally, you can update the description of this table in the **Description** field. For example, you might enter ″Table name for HTTP session data.″
8. Click **Apply** > **Save**.

When the product is restarted, the Session Management facility creates the new SESSIONS table in the specified tablespace.

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 data source from which session management obtains database connections. For example, if the JNDI name of the datasource is ″jdbc/sessions″, specify ″jdbc/sessions.″ The data source represents a pool of database connections and a configuration for that pool (such as the pool size). The data source 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 multirow 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.

Programming issue	Application scenario
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 memory-to-memory replication for the peer-to-peer mode (default memory-to-memory replication)

To configure the session management facility for memory-to-memory session replication for peer-to-peer functions (both client and server function, or both mode) with a single replica, complete the following steps:

1. Create an application cluster. This cluster is used to deploy the application.
   a. Go to the Server Cluster page. Click **Servers**> **Clusters**.
   b. Click **New**.
   c. Type a cluster name for this application cluster.
   d. Define a replication domain. Select the **Create a replication domain for this cluster** check box.
   e. Click **Next**.
   f. Define each cluster member server. Type a cluster member name.
   g. Click **Apply**. Repeat steps **f** through **g** for each server created in this cluster.
   h. Click **Next** and review the summary of changes.
   i. Click **Finish** to complete the configuration.

   You have now created a cluster that contains the deployed application and the replication domain.
2. Enable memory-to-memory session replication for each server.
   a. Go to the appropriate level of session management for the Web container level. Click **Servers** > **Application servers**> *server_name*> **Container Settings**> **Web Container Settings**> **Session management**
   b. Click **Distributed environment settings** under Additional Properties.
   c. Click **Memory-to-memory replication**.
   d. Select the **Replication domain** that you want to use for the replication of sessions.
   e. Select the both client and server **Replication mode**. You must configure all session managers connected to a replication domain to have the same topology. If one session manager instance in a domain is configured to use the client/server topology, then the rest of the session manager instances in that domain must be a combination of servers configured as ″Client only″ and ″Server only″. If one session manager instance is configured to use the peer-to-peer topology, then all session manager instances must be configured as ″Both client and server″.
   f. Click **OK** on the Memory-to-memory replication page.
   g. Optional: If you want to change the tuning parameters, click **Custom tuning parameters** and select a setting or customize one. Click **OK**. Click **Save**.

      Note: Using the default tuning parameter custom settings, which specifies time based write interval of 10 seconds, may result in data loss when an application server in your cluster fails. However, this is just a small opportunity for lost data when compared to the significant improvement in performance.
   h. Click **OK** the Distributed environment settings page.

i. Click **OK** the Session management page.
j. Repeat **a** through **i** for each server.

Memory-to-memory replication settings

Use this page to configure memory-to-memory sessions.

To view the Memory-to-memory sessions page, click **Servers** > **Application servers** > *server_name* > **Web container settings** > **Web container** > **Session management** > **Distributed environment settings** > **Memory-to-memory replication**.

Replication domain:

Specifies the replication domain in which HTTP sessions are replicated.

Replication mode:

Select the mode in which this server has to run: Both, Client and Server. The mode implies whether data is only sent (client), only received (server), or both. The default is both.

Configuring memory-to-memory replication for the client/server mode

To configure the session management facility for memory-to-memory replication using the clients/server mode, complete the following steps:

1. Create an application cluster. This cluster is used to deploy the application.
   a. Go to the Server Cluster page. Click **Servers**> **Clusters**.
   b. Click **New**.
   c. Type a cluster name for this application cluster.
   d. Click **Next**.
   e. Define each cluster member server. Type a cluster member name.
   f. Click **Apply**. Complete steps **e** and **f** for each server created in this cluster.
   g. Click **Next** and review the summary of changes.
   h. Click **Finish** to complete the configuration.

   Do not create a replication domain for the application cluster. You have now created a cluster that contains the deployed application.
2. Create a cluster of session manager replication servers (backup cluster).
   a. Go to the Server Cluster page. Click **Servers**> **Clusters**.
   b. Click **New**.
   c. Type a cluster name for the cluster of session manager replication servers.
   d. Define a replication domain. Select the **Create a replication domain for this cluster** check box.
   e. Click **Next**.
   f. Define each cluster member server. Type a cluster member name.
   g. Click **Apply**. Complete steps **f** through **g** for each server created in this cluster.
   h. Click **Next** and review the summary of changes.
   i. Click **Finish** to complete the configuration.

   This step creates a cluster of backup session manager replication servers and associates a replication domain with that cluster.
3. Enable memory-to-memory session replication for each cluster member server in the application cluster.

a. Go to the appropriate level of session management for the Web container level. Click **Servers** > **Application Servers**> *server_name*> **Container Settings**> **Web Container Settings**> **Session management**
b. Click **Distributed Environment Settings** under Additional Properties.
c. Click **Memory-to-memory replication**.
d. Select the **Replication domain** that you want to use for the replication of sessions.
e. Select the Client only **Replication mode**. You must configure all session managers connected to a replication domain to have the same topology. If one session manager instance in a domain is configured to use the client/server topology, then the rest of the session manager instances in that domain must be a combination of servers configured as Client only and Server only. If one session manager instance is configured to use the peer-to-peer topology, then all session manager instances must be configured as Both client and server. Alternatively, if one DRS Instance is configured in the client only mode then all DRS Instances in the domain must be configured in either the client only or the server only modes.
f. Click **OK** on the Memory-to-memory replication page.
g. Optional: If you want to change the tuning parameters, click **Custom tuning parameters** and select a setting or customize one. Click **OK**. Click **Save**.

   Note: Using the default tuning parameter custom settings, which specifies time based write interval of 10 seconds, may result in data loss when an application server in your cluster fails. However, this is just a small opportunity for lost data when compared to the significant improvement in performance.
h. Click **OK** the Distributed environment settings page.
i. Click **OK** the Session management page.
j. Repeat **a** through **i** for each server.

4. Enable memory-to-memory session replication for each cluster member server in the replication cluster.
   a. Go to the appropriate level of session management for the Web container level. Click **Servers** > **Application Servers**> *server_name*> **Container Settings**> **Web Container Settings**> **Session management**
   b. Click **Distributed Environment Settings** under Additional Properties.
   c. Click **Memory-to-memory replication**.
   d. Select the **Replication domain** that you want to use for the replication of sessions.
   e. Select the Server only **Replication mode**. You must configure all session managers connected to a replication domain to have the same topology. If one session manager instance in a domain is configured to use the client/server topology, then the rest of the session manager instances in that domain must be a combination of servers configured as Client only and Server only. If one session manager instance is configured to use the peer-to-peer topology, then all session manager instances must be configured as Both client and server. Alternatively, if one DRS Instance is configured in the client only mode then all DRS Instances in the domain must be configured in either the client only or the server only modes.
   f. Click **OK** on the Memory-to-memory replication page.
   g. Optional: If you want to change the tuning parameters, click **Custom tuning parameters** and select a setting or customize one. Click **OK**. Click **Save**.

      Note: Using the default tuning parameter custom settings, which specifies time based write interval of 10 seconds, may result in data loss when an application server in your cluster fails. However, this is just a small opportunity for lost data when compared to the significant improvement in performance.
   h. Click **OK** the Distributed environment settings page.
   i. Click **OK** the Session management page.

j. Repeat **a** through **i** for each server.

Classifying WebSphere transaction workload for WLM

This topic describes how to use transaction classes to classify client workload for workload management. The workload is different WebSphere transactions targeted to separate servant regions, each with goals defined by appropriate service classes. Each transaction is dispatched in its own WLM enclave in a servant region process, and is managed according to the goals of its service class.

This topic describes steps to classify transaction workload as a way of managing the workload service objectives. You also need to define the service objectives (goals) for the service classes used. In addition, you must define the service objectives of the WebSphere Application Server for z/OS servers and your business application servers.

For more information about defining service objectives (goals) for each service class, see the *z/OS MVS Planning: Workload Management* book, SA22-7602, for example at http://publibz.boulder.ibm.com/epubs/pdf/iea2w131.pdf, or the z/OS WLM Web page at http://www.ibm.com/servers/eserver/zseries/zos/wlm/.

You can classify your WebSphere work using the WLM CB-type classification criteria:
- Server name (CN)
- Server instance name (SI)
- User ID assigned to the transaction (UI)
- Transaction class (TC)

Note: To get started, you do not need to define special classification rules and work qualifiers, but you may want to do this for your production system.

To classify work using server and userid criteria, you use a combination of the WLM Workload Classification rules in the WLM ISPF dialog panels. For more information about defining WLM Classification rules, see ″Workload management (WLM) for z/OS″ and its related article that includes an example of classification rules.

To classify work using transaction classes, you define and use transaction class mappings, as described in this task. The following steps are used to classify work using transaction classes:

1. Define transaction class mappings based on the HTTP virtual host name, port number, and URI (Universal Resource Identifier - encoded address for any resource on the Web) provided with each work HTTP or HTTPS request.
   a. Create a Transaction Class mapping file (as a simple text file). For example:
   `/wasconfig/t5was/MyTrMapFile.txt`
   b. Edit the Transaction Class mapping file to define each transaction class mapping that you want to use. Define each mapping on a separate line, using the following syntax:

   ```
   TransClassMap host:port uritemplate tclass
   ```

   Note: In the host or port fields, you can use wildcard characters only for the entire field as shown in the following example.

   This syntax is the same syntax as for WebSphere Application Server for z/OS Version 4.0.1. For more information about this syntax, see Transaction class mapping file entries. For example:

   ```
   TransClassMap wsc4.washington.ibm.com:9080  /MyIVT/index.*    TCLMYIVT
   TransClassMap wsc4.washington.ibm.com:9080  /MyIVT/ivtejb     TCLMYEJB
   TransClassMap wsc4.washington.ibm.com:*     /SuperSnoop*     TCLSNOOP
   TransClassMap wsc4.washington.ibm.com:*     /ssb/*           TCLSSB
   TransClassMap *:*                           /admin*          TCLADMIN
   ```
2. Specify the Transaction Class mapping file on the administrative properties for each server that is to handle work classified by transaction class. To specify the Transaction Class mapping file for a server, use the administrative console to complete the following steps:

a. In the navigation pane, click **Servers > Application Servers**.
b. In the content pane, select the server instance, *server_name*.
c. In the Additional Properties list in the contents pane, select **Web Container**.
d. In the Additional Properties list for the Web container, select **Advanced Settings**.
e. In the **Transaction Class Mapping** field, type the name of the Transaction Class mapping file that you edited in an earlier step. For example: `/wasconfig/t5was/MyTrMapFile.txt`

   This sets the following variable in the server's was.env file:

   ```
   protocol_http_transport_class_mapping_file=/wasconfig/t5was/MyTrMapFile.txt
   ```
f. If you want to use a transaction class to classify outbound data that is delivered in response to HTTP and HTTPS requests, select the TCLASS option in the **Network QoS** field. If you specify TCLASS, WebSphere Application Server for z/OS uses the transaction class value that was used to classify the inbound request to the z/OS Workload Manager.

The following table shows classification rules for CB-type work in which the default service class is WSMED and has a reporting class of RWSDEFLT. Work run in the WSPROD WebSphere server is classified as WSMED with a reporting class of RWSPROD, unless it has a transaction class of TCLASS1, TCLASS2, or TCLASS2 assigned through the transaction class mapping file below.

```
Qualifier    Qualifier Start       Service  Report
# type       name      position    Class    Class
- ---------  --------  --------    -------- --------
                         Default: WSMED    RWSDEFLT
1 CN         WSPROD    1           WSMED    RWSPROD
2 . TC       . TCLASS1             WSFAST   RWSPRD1
2 . TC       . TCLASS2             WSMED    RWSPRD2
2 . TC       . TCLASS5             WSSLOW   RWSPRD5
1 CN         WSTEST    1           WSSLOW   RTSTEST
2 . UI       . USER1               WSMED    RTSTSTU2
2 . TC       . TCLASS5             WSSLOW   RTSTST5
```

The following table shows how work can be assigned a transaction class based on its host name, port number, or URI. For example, a web request of http://ibm.com:80/Webap1/myservlet handled by the WSPROD server would be assigned a transaction class of TCLASS1, a service class of WSFAST, and a reporting class of RWSPRD1 by the classification rules shown above.

```
TransClassMap www.ibm.com:80 /Webap1/myservlet TCLASS1
TransClassMap www.ibm.com:* /Webap1/myservlet TCLASS2
TransClassMap *:443 * TCLASS3
TransClassMap *:* /Webap1/myservlet TCLASS4
TransClassMap www.ibm.com:* /Webap5/* TCLASS5
TransClassMap * * TCLASS6
```

Controller and Servant WLM classifications

You should classify the WebSphere Application servant regions to a high STC importance service class so that they can be initialized quickly when WLM determines they are needed. The service class chosen also determines the WLM goal when Java Garbage Collection (GC) is running, which can be CPU intensive. You do not want to set a servant higher in the service class hierarchy than more important work such as production WebSphere, CICS, or IMS transaction servers.

WebSphere application controller regions do some processing to receive work into the system, manage the HTTP transport handler, classify the work and do other housekeeping tasks. Therefore, controller regions should also be classified in SYSSTC or a high importance and velocity goal.

Here is a simple example of the WLM Classification Rules for STC-type work that covers the controller and servant regions started tasks:

```
          --------Qualifier--------           -------Class--------
Action    Type     Name     Start             Service     Report
                                  DEFAULTS: OPS_DEF     ________
```

```
_____ 1  TN      %%DMN    ___              OPS_HIGH    RWSDMN _____  1  TN      T5SRV*   ___              OPS_MED
_____ 1  TN      WS%%%%   ___              SYSSTC      RWSCTLR
 ____ 1  TN      WS%%%%S  ___              OPS_HIGH    RWSSRVR
```

EJB applications

Task overview: Using enterprise beans in applications

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 533.
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 the assembly tool. This process includes setting security. See ″Securing enterprise bean applications″ in the information center. 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. See ″Setting the run time for batched commands with JVM arguments″ and ″Setting the run time for deferred create with JVM arguments″ in the information center.
7. Deploy the application in an application server.
8. Test the modules.
   - As needed, debug problems with the container. See ″Enterprise bean and EJB container troubleshooting tips″ in the information center.
   - Debug access problems. ″See Cannot access an enterprise bean from a servlet, a JSP file, a stand-alone program, or another client″ in the information center.
9. Assemble the production application using 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 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 ″Resources for learning.″

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 ″Resources for learning.″

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.

Between transactions, the state of an entity bean can be cached. The EJB container supports option A, B, and C caching.

- With option A caching, the application server assumes that the entity bean is used within a single container. Clients of that bean must direct their requests to the bean instance within that container. The entity bean has exclusive access to the underlying database, which means that the bean cannot be cloned or participate in workload management if option A caching is used.

  If you intend to use read-only scenarios, WebSphere Application Server provides an alternate, higher-performance variation of option A entity beans. This caching option is called *Multithreaded Read-Only*. Similar to standard option A behavior, the EJB container continues to activate the bean just once and leave it active until the EJB container needs space in its active instance cache. However, the EJB container differs from standard option A in the following behaviors:
  - It reloads the state of the bean from persistent storage periodically in response to the user invoking a method on it to pick up any changes that may have been made to the persistent store since the last time the bean was loaded. You can configure this function through a *Reload Interval* setting in the bean‘s deployment descriptor. For more information, see ″Developing read-only entity beans″ in the information center.
  - The state of the bean is not written to persistent store by the EJB container at the end of the transaction, nor is the bean's ejbStore() method be invoked.
  - The EJB container permits method invocations from more than one client (thread) on the same bean instance. This differs from the standard EJB component for the internals of a bean. You must keep this aspect in mind when developing your bean, and ensure that any logic in the bean‘s business methods is overall thread-safe.
- With option B caching, the entity bean remains active in the cache throughout the transaction but is reloaded at the start of each method call.
- With option C caching (the default), the entity bean is always reloaded from the database at the beginning of each transaction. A client can attempt to access the bean and start a new transaction on any container that has been configured to host that bean. This is similar to the session clustering facility described for HTTP sessions in that the entity bean's state is maintained in a shared database that can be accessed from any server when required.

This product supports the cloning of stateful session bean home objects among multiple application servers. However, it does not support the cloning of a specific instance of a stateful session bean. Each instance of a particular stateful session bean can exist in just one application server and can be accessed only by directing requests to that particular application server. State information for a stateful session bean cannot be maintained across multiple members of a server cluster. However, enabling stateful session bean failover and configuring the EJB container to use memory-to-memory replication does enable stateful session bean failover to be replicated to other servers in the cluster so that failover can occur to the backup server if the primary server for a stateful session bean stops for some reason. For more information about stateful session bean failover, see “Stateful session bean failover for the EJB container” on page 187.

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 187.

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 ″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.

View links to additional information about:
- Planning, business scenarios, and IT architecture
- Programming model and decisions
- Programming instructions and examples
- Programming specifications

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
- Rules and Patterns for Session Facades

  EJB programming practice: Fronting entity beans with a session-bean facade.
- 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.
- Java™ 2 Platform: Compatibility with Previous Releases

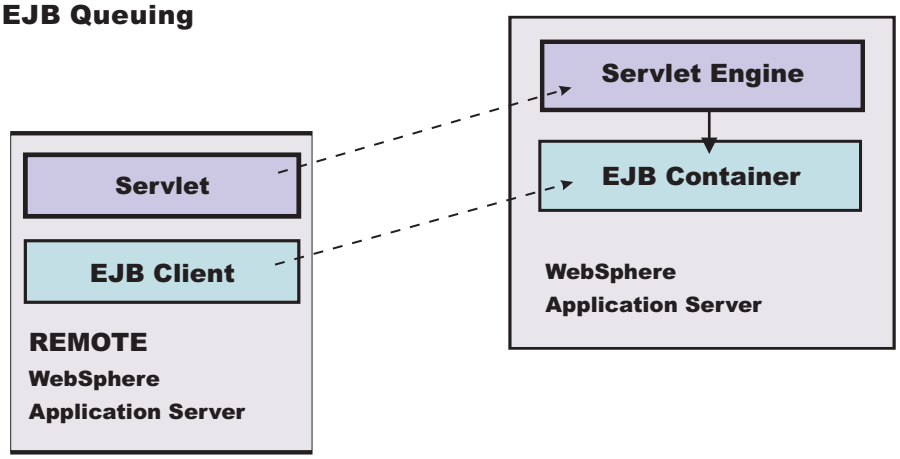
  This Sun Microsystems article includes both source and binary compatibility issues.

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.

EJB Queuing



Using access intent policies

You can use access intent policies to help the product run-time 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) . You can also create your own custom policies.

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.
3. Create a custom access intent policy. See ″Creating a custom access intent policy″
4. Apply access intent policies to BMP entity bean methods by using the AccessIntent API.
5. Apply multiple access intent policies to methods by using application profiling.

Applying access intent policies to beans

You apply an access intent policy to an application’s entity beans through the assembly tools.

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 - See ″Starting an assembly tool″ in the information center.
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

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.

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. Nullable columns are automatically excluded from overqualified updates at deployment time; concurrent changes to a nullable field might result in lost updates. When used with the IBM Rational Application Developer product, this product provides support for selecting a subset of the non-nullable columns that are to be reflected in the overqualified update statement that is generated in the deployment code to support optimistic policies.

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.

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 are most likely to be needed next by an application. A *read-ahead hint* is a representation of the related beans that are to be read. It 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 CMRs 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 their 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*), a bean's data columns appear only once, at the position it first appears 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 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 a relationship *employees* with the Employee bean and also has a relationship *manager* 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.

Some things to consider

When developing your read-ahead hints, you should keep the following in mind:
- Read ahead on long or complex paths can result in a query that is too complex to be useful. Read ahead on root or leaf inheritance mappings need particular care. You should add up the number of tables that are involved in the preload and then consider whether a join that complex is really a reasonable query on your target database.
- Read ahead does NOT work in the following cases:
  - preload paths across M:N relationships
  - preload paths across recursive enterprise bean relationships or recursive fk relationships
  - where multiple instances of the same table occur on the same path (whether through a recursive relationship or not).
  - when readAhead 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.

Using access intent policies to avoid 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) then the entity bean is updated within the same transaction, this causes a lock upgrade (to 'Exclusive').

If this scenario occurs on multiple threads concurrently, then a deadlock can happen. This is because multiple 'Read' locks can be obtained concurrently, but only one 'Exclusive' lock can be obtained only when all other locks have been dropped. This one 'Exclusive' lock can never be obtained in this scenario, because all transactions are attempting the lock upgrade.

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 173.

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. See ″ Starting an assembly tool″ in the information center.
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 has not changed since the last load (with proper locking based on access intent's concurrency control attribute.)

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:
Usage Scenario

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.

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.
- User1 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.
- User3 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.
- User1 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.
- User3 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.
- User1 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 run-time 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 officially 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 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.

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. 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 (a feature of WebSphere Business Integration Server Foundation).

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 run-time 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 tools.

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) ) {
     boolean 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.

Access intent exceptions

The following exceptions are thrown in response to the application of access intent policies:

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 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

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.

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. Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction.

Access intent best practices

This topic outlines issues to consider when applying access intent policies to Enterprise JavaBeans (EJB) methods.

- **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

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 databases, the weakest lock is a shared one, and the query runs without a USE AND KEEP UPDATE LOCKS clause. On Oracle databases, however, the weakest lock is an update lock; this means that the SQL query must contain a USE AND KEEP UPDATE LOCKS clause. However, not every SQL statement necessarily supports USE AND KEEP UPDATE LOCKS; for example, if the query is being run against multiple tables in a join, USE AND KEEP UPDATE LOCKS is not supported. To avoid this problem, try either of the following:

- Modify your SQL query or reconfigure your application so that an update lock is supported
- 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.

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 2 147 483 674

Default datasource 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,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

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 helps to speed up the application start time.

EJB types can, however, 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.

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

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 runtime. This helps to speed up the application start time.

All EJB types within a server can, however, 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.

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. Select **Servant** process type.
8. In the Additional Properties area, select **Java Virtual Machine**.
9. In the Additional Properties area, select **Custom Properties**.
10. Select the **New** box.
11. In the **Name** entry field, type *com.ibm.websphere.ejbcontainer.initializeEJBsAtStartup*.
12. 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").
13. Select **OK**.

Stateful session bean failover for the EJB container

WebSphere Application Server Version 6.0 enables you to construct applications with the assumption that your applications using stateful session beans are not limited by unexpected server failures. This version of the product utilizes the functions of the Data Replication Service (DRS) and Workload Management (WLM) so you can enable stateful session bean failover.

Because you might not want to enable failover for every single stateful session bean installed in the EJB container, you can override the EJB container settings at either the application or EJB module level. You can either enable or disable failover at each of these levels. For example, consider the following situations:

- You want to enable failover for all applications except for a single application. To do this, you enable failover at the EJB container level and override the setting at the application level to disable failover on the single application.
- You want to enable failover for a single installed application. To do this, disable failover at the EJB container level and then override the setting at the application level to enable failover on the single application.
- You want to enable failover for all applications except for a single module of an application. To do this, enable failover at the EJB container level, then override the setting at the module application level to disable failover on the single module.
- You want to enable failover for a single installed EJB module. To do this, disable failover at the EJB container level and then override the setting at the EJB module level to enable failover on the single EJB module.

For information about enabling stateful session bean failover from the administrative console, see "Enabling or disabling stateful session bean failover with the EJB container panel" on page 192, "Enabling or disabling stateful session bean failover with the enterprise applications panel" on page 192, and "Enabling or disabling stateful session bean failover with the EJB modules panel" on page 193.

Stateful session bean activation policy with failover enabled

WebSphere Application Server enables an application assembler to specify an activation policy to use for stateful session beans. It is important to consider that the only time the EJB container prepares for failover (by replicating the stateful session bean data using DRS) is when the stateful session bean is passivated. If you configure the bean with an *activate once* policy, the bean is essentially never passivated. If you configure the *activate at transaction boundary* policy, the bean is passivated whenever the transaction that the bean is enlisted in completes. For stateful session bean failover to be useful, the activate at transaction boundary policy is required.

Rather than forcing you to edit the deployment descriptor of every stateful session bean and reinstall the bean, the EJB container simply ignores the configured activation policy for the bean when you enable failover. The container automatically uses the activate at transaction boundary policy.

Stateful session bean use of container managed units of work or bean managed units of work with failover enabled

The relevant "units of work" in this case are *transactions* and *activity sections*. The product supports stateful session bean failover for container managed transactions (CMT), bean managed transactions (BMT), container managed activity sessions (CMAS), and bean managed activity sections (BMAS). However, in the container managed cases, preparation for failover only occurs if trying to send a request for an enterprise bean method invocation results in no connection to the server. Also, if the server fails *after* a request is sent to it and acknowledged, failover does not occur. When a failure occurs in the middle of a request or unit of work, WLM cannot safely fail over to another server without some compensation code being executed by the application. When that happens, the application receives a Common Object Request Broker Architecture (CORBA) exception and minor code telling it that transparent failover could not occur because the failure happened during execution of a unit of work. The application should be written to check for the CORBA exception and minor code, and compensate for the failure. After the compensation code executes, the application can retry the requests and if a path exists to a backup server WLM routes the new request to a new primary server for the stateful session bean.

The same is true for bean managed units of work (transactions or activity sessions). However, bean managed work introduces a new possibility that needs to be considered.

For bean managed units of work, the failover process is not always able to detect that a BMT or BMAS started by a stateful session bean method has not completed. Thus, it is possible that failover to a new server can occur despite the unit of work failing during the middle of a transaction or session. Because the unit of work is implicitly rolled back, WLM behaves as if it is safe to transparently fail over to another server, when in fact some compensation code might be required. When this happens, the EJB container detects this on the new server and initiates an exception. This exception occurs under the following scenario:

1. A method of a stateful session bean using bean managed transaction or activity session calls begin on a UserTransaction it obtained from the SessionContext. The method does some work in the started unit of work, but does not complete the transaction or session before returning to the caller of the method.
2. During post invocation of the method started in step 1, the EJB container suspends the work started by the method. This is the action required by Enterprise JavaBeans specification for bean managed units of work when the bean is a stateful session bean.
3. The client starts several other methods on the stateful session bean. Each invocation causes the EJB container to resume the suspended transaction or activity session, dispatch the method invocation, and then suspend the work again before returning to the caller.
4. The client calls a method on the stateful session bean that completes the transaction or session started in step 1.

This scenario depicts a *sticky* bean managed unit of work. The transaction or activity session sticks around for more than a single stateful session bean method. If an application uses a sticky BMT or BMAS, and the server fails after a sticky unit of work completes and before another sticky unit of work starts, failover is successful. However, if the server fails *before* a sticky transaction or activity session completes, the failover is not successful. Instead, when the failover process routes the stateful session bean request to a new server, the EJB container detects that the failure occurred during an active sticky transaction or activity session. At that time, the EJB container initiates an exception.

Essentially, this means that failover for both container managed and bean managed units of work is not successful if the transaction or activity session is still active. The only real difference is the exception that occurs.

Application Design Considerations

You should consider the following when designing applications that use the stateful session bean failover process:

- To avoid the possibility described in the section above, you are encouraged to write your application to configure stateful session beans to use container managed transactions (CMT) rather than bean managed transactions (BMT).
- If you desire immediate failover, and your application creates either an HTTP session or a stateful session bean that stores a reference to another stateful session bean, then the administrator must ensure the HTTP session and stateful session bean are configured to use the same data replication service (DRS) replication domain.
- Do Not use a local and a remote reference to the same stateful session bean.

  Normally a stateful session bean instance with a given primary key can only exist on a single server at any given moment in time. Failover might cause the bean to be moved from one server to another, but it never exists on more than one server at a time. However, there are some unlikely scenarios that can result in the same bean instance (same primary key) existing on more than one server concurrently. When that happens, each copy of the bean is unaware of the other and no synchronization occurs between the two instances to ensure they have the same state data. Thus, your application receives unpredictable results.

  Attention: To be sure to avoid this situation you must remember that with failover enabled, your application should **never** get both a local (EJBLocalObject) and remote (EJBObject) reference to the same stateful session bean instance.

For z/OS users only

Stateful session bean failover on WebSphere Application Server Network Deployment for z/OS is slightly different than that on the WebSphere Application Server Network Deployment product. In addition to the failover solution discussed here, z/OS users can also enable failover among servants in an unmanaged server. For more information, refer to:

- "Peer recovery of transactions" on page 1308
- Setting up peer restart and recovery
- Considerations for clustered servers and stateful session beans

Because the z/OS product has a control region and servant regions and the Network Deployment product does not, there is one failover scenario that is unique to z/OS. That is failover from one servant region to another servant region (loss of a servant without loss of the controller).

Customers currently using the HFS-based technique on z/OS will likely want to continue with that choice.

In an unmanaged z/OS server, stateful session bean failover among servants can be enabled. Failover only occurs between the servants of a given unmanaged server. If an unmanaged z/OS server has only one servant, then enabling failover has no effect. An unmanaged z/OS server that has failover enabled does not fail over to another unmanaged z/OS server. To enable failover in an unmanaged server, refer to

Stateful session beans failover settings (applications):

Each Enterprise JavaBeans container provides a method for stateful session beans to fail over to other servers. This enables you to specify whether failover occurs for the stateful session beans in this module. You can also override the parent object's stateful session bean replication settings for this module.

To view this administrative console page, click **Applications > Enterprise Applications >** *application* **> Stateful Session Bean Failover Settings**.

Note: These settings are ignored for 5.x application targets.

Enable stateful session bean failover using memory to memory replication:

Specifies whether the EJB Container attempts failover for all of the stateful session beans in this application.

This checkbox overrides the default stateful session bean failover setting that the administrator configured for an EJB container. De-selecting this checkbox disables failover for this application.

Data type	Checkbox
Range	Selected or unselected.

Use replication settings from the EJB container:

Specifies that the replication settings configured for the EJB container are used for this application. If you select this option and you want the application to use stateful session bean failover, you must define memory to memory replication for the EJB container on each server you want to use failover.

Data type	Radio button
Range	Selected or unselected.

Use application replication settings:

Specifies that the replication settings configured for this application are used for memory to memory replication of the stateful session bean data.

If you select this button, you override the EJB container settings. This button 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 application.

Data type	Radio button
Range	Selected or unselected.

Stateful session beans failover settings (EJB modules):

Each Enterprise JavaBeans container provides a method for stateful session beans to fail over to other servers. This enables you to specify whether failover occurs for the stateful session beans in this module. You can also override the parent object's stateful session bean replication settings for this module.

To view this administrative console page, click **Applications > Enterprise Applications >** *application* **> EJB modules** > *.jar* > **Stateful session bean failover settings**.

Note: These settings are ignored for 5.x application targets.

Enable stateful session bean failover using memory to memory replication:

Specifies whether the EJB Container attempts failover for all of the stateful session beans in this module.

This checkbox overrides the default stateful session bean failover setting that the administrator configured for an EJB container or the module's application. De-selecting this checkbox disables failover for this module.

Data type	Checkbox
Range	Selected or unselected.

Use application or EJB container replication settings:

Specifies that the replication settings configured for the EJB container or application are used for this module. If you select this option and you want the application to use stateful session bean failover, you must define memory to memory replication for the EJB container on each server you want to use failover.

Data type	Radio button
Range	Selected or unselected.

Use EJB module replication settings:

Specifies that the replication settings configured for this EJB module are used for memory to memory replication of the stateful session bean data.

If you select this button, you override the replication settings for the EJB container and application. This button 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	Radio button
Range	Selected or unselected.

Enabling failover of servants in an unmanaged server:

In an unmanaged z/OS server, stateful session bean failover among servants can be enabled. Failover only occurs between the servants of a given unmanaged server. If an unmanaged z/OS server has only one servant, then enabling failover has no effect. An unmanaged z/OS server that has failover enabled does not fail over to another unmanaged z/OS server.

Failover in an unmanaged server is enabled by setting a JVM property.

1. Start the administration console.
2. Select **Servers**.
3. Select **Application Servers**.
4. Select the server that you want to modify.
5. In the Server Infrastructure area, select **Java and Process Management**.
6. Select **Process Definition**.
7. Select **Servant**.
8. Under Additional Properties, select **Java Virtual Machine**.
9. Under Additional Properties, select **Custom Properties**.
10. Click on the **New** button.
11. In the Name input field, type **EJBContainerEnableUnmanagedServerReplication**.
12. In the Value input field, type **true**.
13. Click **Apply**.
14. Click **OK**.

Enabling or disabling stateful session bean failover with the EJB container panel

You can use the EJB Container administrative console panel to enable stateful session bean failover. Selecting the checkbox on the panel enables you to configure whether the failover is active for all stateful session beans in the specified EJB container.

1. Start the administrative console.
2. Select **Servers**.
3. Select **Application Servers**.
4. Select the server you want to work with.
5. Select **EJB Container Settings**.
6. Select **EJB container**.
7. Check the box labeled **Enable stateful session bean failover using memory-to-memory replication**. 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.
8. Select **OK**.

Enabling or disabling stateful session bean failover with the enterprise applications panel

You can use the enterprise applications administrative console panel to enable or disable stateful session bean failover for all stateful session beans in the specified application.

1. Start the administrative console.

2. Select **Applications**.
3. Select **Enterprise Applications**.
4. Select the application you want to work with.
5. Under Additional Properties, select **Stateful Session Bean Failover Settings**. The Stateful Session Bean Failover Settings panel appears.
6. Select **Enable stateful session bean failover using memory to memory replication**. This enables failover for all stateful session beans in this application. If you want to *disable* the failover, clear this checkbox.
7. Select your choice of **Replication settings**. You have a choice of two radio buttons:

   Use replication settings from EJB container
   If you select this button, any replication settings defined for this application are ignored.

   Attention: Note to the system administrator: if you use this radio button, then memory to memory replication must be configured at the EJB container level. Otherwise, the settings on this panel are ignored by EJB container during server startup and the EJB container will log a message indicating that stateful session bean failover is not enabled for this application.

   Use application replication settings
   If you select this button, you override the EJB container settings. This button 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 application.
8. Select **OK**.

Enabling or disabling stateful session bean failover with the EJB modules panel

You can use the enterprise applications administrative console panel to enable or disable stateful session bean failover for all stateful session beans in the specified module.

1. Start the administrative console.
2. Select **Applications**.
3. Select **Enterprise Applications**.
4. Select the application you want to work with.
5. Under Related items, select **EJB modules**.
6. Select the *.jar* file you want to work with.
7. Select **Stateful session bean failover settings**.
8. Select **Enable stateful session bean failover using memory to memory replication**.
9. Select your choice of **Replication settings**. You have a choice of two radio buttons:

   Use application or EJB container replication settings
   If you select this button, any replication settings defined for this EJB module are ignored.

   Attention: Note to the system administrator: if you use this radio button, then memory to memory replication must be configured at the EJB container level. Otherwise, the settings on this panel are ignored by EJB container during server startup and the EJB container will log a message indicating that stateful session bean failover is not enabled for this application.

   Use EJB module replication settings
   If you select this button, you override the replication settings for the EJB container and application. This button 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.

10. Select **OK**.

EJB cache settings

Use this page to configure and manage the cache for a specific EJB container. To 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.

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 2 147 483 674
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.

The product uses interoperable versions of some class types to enable interoperability. However, older 4.0.x client and application server versions do not support the interoperability classes, which makes them uninteroperable with versions that use the classes. The system property *com.ibm.websphere.container.portable* remedies this situation by enabling newer versions of the

application server to turn off the interoperability classes. This lets a more recent application server return class types that are interoperable with an older client.

Depending on the value of com.ibm.websphere.container.portable, application servers at versions 5 and later, and 4.0.3 and later, return different classes for the following:
- Enumerations and collections returned by EJB 1.1 finder methods
- EJBMetaData
- Handles to:
  - Entity beans
  - Session beans
  - Home interfaces

If the property is set to `false`, application servers return the old class types, to enable interoperability with 4.0.2 and earlier. If the property is set to `true`, application servers return the new classes.

The following tables show interoperability characteristics for various version combinations of application servers and clients as well as default property values for each combination.

Interoperability of Version 4.0.x client with Version 5 (and later) application server

Ideally, all 4.0.x clients that use Version 5 or later application servers should be at Version 4.0.3 or later.

Version 5 and later application servers return the interoperability class types by default (`true`). This can cause interoperability problems for distributed clients at versions 4.0.1 or 4.0.2. In particular, problems can occur with collections and enumerations returned by Enterprise JavaBeans Version 1.1 finder methods.

Although it is strongly discouraged, you can set com.ibm.websphere.container.portable to `false` on a Version 5 and later application server. This causes the application server to return the old class types, providing interoperability with clients at Version 4.0.2 and earlier. This is discouraged because:
- The Version 5 application server instance would become non-J2EE 1.3 (and later) compliant with regard to handles, home interface handles, and EJBMetaData.
- EJB 1.x finder methods return collection and enumeration objects that do not originate from ejbportable.jar.
- Interoperability restrictions still exist with the property set to `false`.
- Version 5 and later client handles to entity beans and home interfaces do not work across domains for the server you set to `false`.

  If you would like to use updated Handle classes in EJB 2.x-compliant beans but have one of the older clients (versions 4.0.2 and earlier) installed, set the system property com.ibm.websphere.container.portable.finder to `false`. With this setting in place, the Version 5 and later application server uses the updated handles but returns the enumerations and collections that were used in the earlier clients.

Interoperability of client at Version 4.0.2 and earlier with Version 5 (and later) application server

Client at Version 4.0.2 and earlier, using this function	Application server at Version 5 and later, property true (default)	Application server at Version 5 and later, property false
EJBMetaData	Does not work	Works for 4.0.2 client
Handle to session bean	Does not work	Works
Handle to entity bean	Does not work	Does not work across cells
Enumeration returned by EJB 1.x finder method	Does not work	Works
Collection returned by EJB 1.x finder method	Does not work	Works
Handle to home interface	Does not work	Does not work across cells

If you would like to use updated Handle classes in EJB 2.x-compliant beans but have one of the older clients (versions 4.0.2 and earlier) installed, set the system property com.ibm.websphere.container.portable.finder to `false`. With this setting in place, the Version 5 and later server uses the new Handle classes but returns the older enumeration and collection classes.

Interoperability of client at Version 4.0.3 and later with Version 5 and later application server

Clients at Version 4.0.3 and later work well with Version 5 and later application servers. However, if you set the com.ibm.websphere.container.portable to `false`, client handles to entity beans and home interfaces do not work across domains for the server you set to `false`.

Client at Version 4.0.3 and later, using this function	Application server at Version 5 and later, property true (default)	Application server at Version 5 and later, property false
EJBMetaData	Works	Works
Handle to session bean	Works	Works
Handle to entity bean	Works	Does not work across cells
Enumeration returned by EJB 1.x finder method	Works	Works
Collection returned by EJB 1.x finder method	Works	Works
Handle to home interface	Works	Does not work across cells

Interoperability of Version 5 and later client with Version 4.0.x application server

Clients at Version 5 and later work well with Version 4.0.3 application servers if you set com.ibm.websphere.container.portable to `true`. Client handles to entity beans and home interfaces do not work across domains for any Version 4.0.3 server with com.ibm.websphere.container.portable at the default value, `false`. Version 5 client handles to application servers at Version 4.0.2 and earlier also have restrictions.

Client at Version 5 and later, using this function	Application server at Version 4.0.3, property true	Application server at Version 4.0.3, property false (default)	Application server at Version 4.0.2 or earlier
EJBMetaData	Works	Works	Works for 4.0.2 server only
Handle to session bean	Works	Works	Works
Handle to entity bean	Works	Does not work across domains	Does not work across domains
Enumeration returned by EJB 1.x finder method	Works	Works	Works
Collection returned by EJB 1.x finder method	Works	Works	Works
Handle to home interface	Works	Does not work across domains	Does not work across domains

Interoperability of zSeries Version 4.0.x client with Version 5 and later application server

The only valid configuration for container interoperability with zSeries Version 4.0.x clients is the default configuration for the Version 5 application server.

Interoperability of Version 5 and later client with zSeries Version 4.0.x application server

Version 5 clients should work with a zSeries Version 4.0.x application server with the correct interoperability fixes described in the zSeries documentation. The interoperability characteristics should be the same as for a Version 4.0.3 distributed application server with the property set to `true`.

Client at Version 5 and later, using this function	zSeries application server at Version 4.0.x
EJBMetaData	Works
Handle to session bean	Works
Handle to entity bean	Works
Enumeration returned by EJB 1.x finder method	Works
Collection returned by EJB 1.x finder method	Works
Handle to home interface	Works

EJB Container Cache tuning

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.

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. See ″Preparing to host applications″ in the information center for additional information.
2. Update the configuration for each EJB module as needed for the deployment environment.
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. See ″Diagnosing problems (using diagnosis tools)″ in the information center.

Troubleshooting tips for EJBDEPLOY relationships

Problems may exist when EJBDeploy creates a data relationship in DB2 for z/OS Version 7.x. EJBDeploy creates a table with a composite of the two primary keys of the EJBs that are related to each other. If the composite keys are larger than 254 characters, DB2 for z/OS V7.x will not accept this relationship and the user will be conforonted with errors such as:

```
DSNT408I SQLCODE = -613, ERROR:  THE PRIMARY KEY OR A UNIQUE CONSTRAINT
IS TOO LONG OR HAS TOO MANY COLUMNS
DSNT418I SQLSTATE   = 54008 SQLSTATE RETURN CODE
```

This problem can be seen when the primary keys that are created for the two related beans have primary keys that are strings. This results in the composite being made up of 2 varchar(250) primary keys for a total of 500, which is greater than 254 maximum in DB2 for z/OS version 7.x.

Things to consider when utilizing top-down mappings to ensure you do not experience this problem:

- Top-down mappings are a guideline and must be reviewed with the DBA.
- Schemas created 'top-down' by EJBDeploy are designed only for testing, and as a guideline for the actual schema required. The use of the 'meet-in-the-middle' mapping does not present this problem.
- The composite key constraint problem is not experienced when using DB2 V8, which has 2K max key lengths.

EJBDEPLOY_JVM_OPTIONS

Set the `EJBDEPLOY_JVM_OPTIONS` property to override Java virtual machine (JVM) options that are passed to the code that deploys EJBs (ejbdeploy.sh). Set this property in one of the following locations: *deploymentmanager*/bin/setupCmdLine.sh or *appServerHome*/bin/setupCmdLine.sh

For example, the following command increases the heap size of the JVM for ejbdeploy.sh:

```
export EJBDEPLOY_JVM_OPTIONS="-Xms128m -Xmx512m"
```

EJB module collection

Use this page to manage the EJB modules deployed in a specific application.

To view this administrative console page, click **Applications > Enterprise Applications >** *applicationName* **> EJB modules**. Click the check boxes to select one or more of the EJB modules in your collection.

Remove: Removes a module from the deployed application. The module is deleted from the application in the WebSphere Application Server configuration repository and also from all 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 then 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 then 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 then the application is restarted.

URI:

When resolved relative to the application URL, this specifies the location of the module's archive contents on a file system. The URI matches the <ejb> or <web> tag in the <module> tag of the application deployment descriptor.

There are three buttons on this panel.

Remove
removes the EJB Module

Remove File
removes the specified file from the EJB Module

Update
updates the module or the application. With Update, you can add, remove, or replace modules

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* **> EJB modules >** *moduleName*.

URI:

When resolved relative to the application URL, this specifies the location of the module archive contents on a file system. The URI must match the URI of a ModuleRef URI in the deployment descriptor of the deployed application (EAR).

Alternate DD:

Specifies a deployment descriptor to be used at run time instead of the one installed in the module.

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

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. See ″Developing ActiveX application client code″ in the information center.
   b. Develop J2EE application client code. See ″Developing J2EE application client code″ in the information center.
   c. Develop pluggable application client code. See ″Developing Pluggable application client code″ in the information center.
   d. Develop thin application client code. See ″Developing Thin application client code″ in the information center.

View the WebSphere Application Server Clients Samples Gallery for more information. To access these samples, install WebSphere Application Server Clients, and retrieve the samples from your local file system as the following command indicates:

<*install_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 run-time environment.

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

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.
- An ActiveX to EJB Bridge run time for ActiveX to EJB Bridge application client applications (only for 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: 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

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″ in the information center..

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″ in the information center.

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 Web 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.

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, 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 run-time 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 section provides some 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 UNIX platforms) 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

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.

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.

For current information available from IBM Support on known problems and their resolution, see the IBM customer 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 customer support page.

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).

This task only applies to J2EE application clients.

1. Open a command window and invoke the following script to launch J2EE application clients using the launchClient shell:

   ```
   install_root/bin/launchClient.bat
   ```

   The launchClient batch command starts the application client run time, which:
   - Initializes the client run time.
   - Loads the class that you designated as the main class with an assembly tool.
   - Runs the main method of the application client program.

   When your program terminates, the application client run time cleans up the environment and the Java virtual machine (JVM) code ends.

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 section describes the Java 2 Platform Enterprise Edition (J2EE) command line syntax for the launchClient tool for WebSphere Application Server. You can use the `launchClient` command from a node within a Network Deployment environment. **However, do not attempt to use the `launchClient` command from the Deployment Manager.**

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:

-CCsoapConnectorPort
The Simple Object Access Protocol (SOAP) connector port. If you do not specify this argument, the WebSphere Application Server default value is used.

-CCverbose
This option displays additional information messages. The default is `false`.

-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.

-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.

-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`.

-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.

-CCinitonly
Use this option to initialize application client run time for ActiveX application clients without launching the client application. The default is `false`.

-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.

-CCpropfile
Indicates the name of a properties file that contains launchClient properties. Specify the properties without the `-CC` prefix in the file. For example: `verbose=true`.

-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 <*install_root*>/ properties/client.policy.

-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>.

-CCexitVM
Use this option to have the WebSphere Application Server call the System.exit() method after the client application completes. The default is false.

-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.

-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.

-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.

The following examples demonstrate correct syntax.

On the Windows operating system:

```
launchClient c:\earfiles\myapp.ear -CCBootstrapHost=myWASServer -CCverbose=true
app_parm1 app_parm2
```

On the UNIX operating system:

```
./launchClient.sh /usr/earfiles/myapp.ear -CCBootstrapHost=myWASServer -CCverbose=true
app_parm1 app_parm2
```

Specifying the directory for an expanded EAR file:

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:

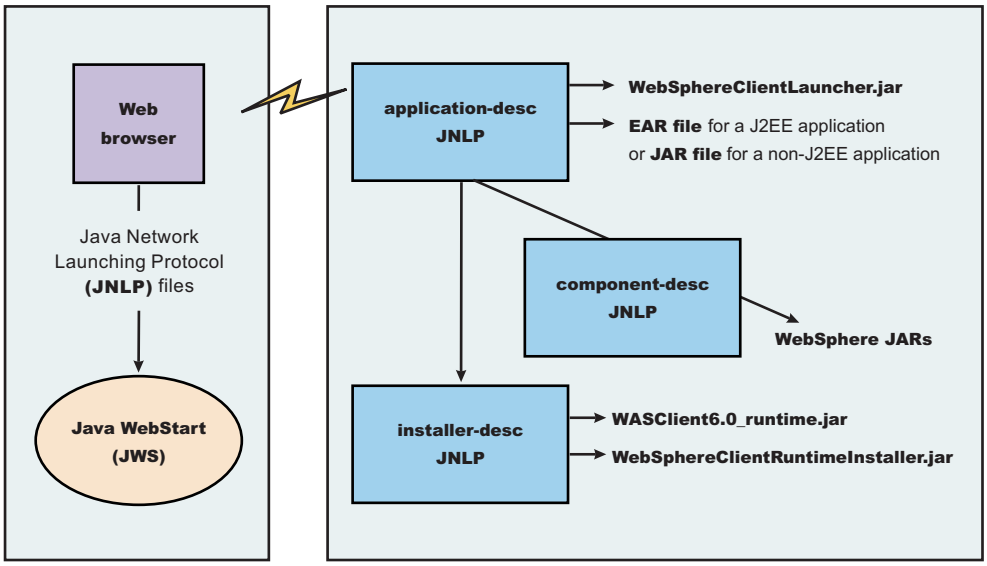
1. Specify a directory to extract the EAR file by setting the `com.ibm.websphere.client.applicationclient.archivedir` Java system property. If the directory does not exist or is empty, the EAR file is extracted normally. If the EAR file was previously extracted, the launchClient tool reuses the directory.
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. 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 product version 1.4.2 or later is highly recommended. The following operating systems support running J2EE application client applications and or Thin application client applications using Java Web Start:

- Red Hat Enterprise Linux for Intel, Version 3.0
- SuSE Linux Enterprise Server, Versions 8 and 9
- Windows 2000 Professional, Windows 2000 Professional, Windows Advance Server, and Windows 2000 Advance Server
- AIX, Versions 5.1, 5.2, and 5.3
- Solaris, Versions 8 and 9
- HP-UX 11i

You can use Java Web Start on the Java 2 Standard Edition Developer Kits that IBM provides, packaged in 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.

Using 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 219
- “Client application Java Network Launcher Protocol deployment descriptor file” on page 221
- “ClientLauncher class” on page 224

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.

1. Prepare the Application Clients run-time dependency component for JWS.
2. Prepare the Application Clients run-time library component for JWS.
3. **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.

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. The client application has an Application Clients run-time dependency that provides the Java 2 Runtime

Environment from IBM, Application Clients run-time properties, the SSL KeyStore and TrustStore file, and the 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="WASclient6.0" href="/WebSphereClientRuntimeWeb/Runtime/jnlp.jsp"/>
```

It 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 which 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 the following example for details:

  ```
  <resources>
  <j2se version="WASclient6.0" href="/WebSphereClientRuntimeWeb/Runtime/jnlp.jsp"/>
  <jar href="Launcher/WebSphereClientLauncher.jar" main="true"/>
  <jar href="lib/j2eeclient.ear"/>
  <property name="com.ibm.websphere.client.launcher.ear" value="j2eeclient.ear"/>
  </resources>
  ```

- 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 download to JWS and the name of the class containing main method entry point is specified in the system property, `com.ibm.websphere.launcher.main`.

  ```
  <resources>
  <j2se version="WASclient6.0" href="/WebSphereClientRuntimeWeb/Runtime/jnlp.jsp"/>
      <extension name="WebSphere Runtime"
                          href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/jnlp.jsp"/>
      <jar href="Launcher/WebSphereClientLauncher.jar" main="true"/>
      <jar href="lib/thinclient.jar"/>
      <property name="com.ibm.websphere.client.launcher.main"
                        value="myapp.sample.thinclient.ThinClientMain"/>

  </resources>
  ```

  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="WebSphere Runtime"
                      href="/WebSphereClientRuntimeWeb/Runtime/WebSphereJars/jnlp.jsp"/>
  ```

  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=${WAS_ROOT}/etc/DummyClientKeyFile.jks </argument>
   <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/DummyClientTrustFile.jks </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` file 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>
   ```

   If any of the default settings in the `sas.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 ${*install_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/DummyClientKeyFile.jks </argument>
   <argument>-CCDcom.ibm.ssl.trustStore=\${WAS_ROOT}/etc/DummyClientTrustFile.jks </argument>
   ```

Here is an example of the client application JNLP descriptor file for a J2EE Application client application.

```
<%--    This is a generic jnlp for a client app.  It will specify the WAS JRE
    as a dependency as well as the client launcher
-->
<%! private final String description="J2EE Client Example";    private final
String earName="J2EEWebStart.ear";
%>
<%  // locally declared variable

String urlSt = request.getRequestURL().toString();
String jnlpCodeBase=urlSt.substring(0,urlSt.lastIndexOf('/'));
String jnlpRefURL=urlSt.substring(urlSt.lastIndexOf('/')+1,urlSt.length());
// The client application descriptor noted a resource reference to be resolved at deploy time as following
%>
<%--
 Need to set a JNLP mime type - if Web Start is installed on the client,
 this header will induce the browser to drive the Web Start Client

--%><%
 response.setContentType("application/x-java-jnlp-file");   1
 response.setHeader("Cache-Control", null);
 response.setHeader("Set-Cookie", null);
 response.setHeader("Vary", null);
%>
<?xml version="1.0" encoding="utf-8"?
<!-- JNLP File for <%=description %> -->
<jnlp
  spec="1.0+"
 <%-- Automate the code base response
-->% codebase="<%=jnlpCodeBase%>"
href="<%=jnlpRefURL%>"
 <information>
  <title><%=description %></title>
  <description kind="short"><%=description %></description>
  <description kind="tooltip"><%=description %></description>
  <offline-allowed></offline-allowed>
 </information>
 <security>
```

```
 <all-permissions></all-permissions>
</security>
 <resources>
   <%-- The URL for the Client JRE installer --%>
    WASclient6.0"
href="/WebSphereClientRuntimeWeb/Runtime/jnlp.jsp"></j2se> 2

  <%-- Specify the client launcher --%>
    <jar href="../Launcher/WebSphereClientLauncher.jar" main="true"> </jar> 3

  <%-- Ear we want to download to the client --%>

   <jar href="<%=earName%>"></jar> 4
    <%-- The launcher depends on this property to be set --%>
    <property name="com.ibm.websphere.client.launcher.ear"
value="<%=earName%>"><property> 5


  <resources>
<%-- Web Start will consider the Launcher as the application to run -->
<application-desc> 6
<argument>-CCproviderURL=corbaloc:iiop:your_server_hostname </argument>  7
<
 <argument>-
CCDcom.ibm.ssl.keyStore=\${install_root}/etc/DummyClientKeyFile.jks</argument>  8

<argument>-
CCDcom.ibm.ssl.trustStore=\${install_root}/etc/DummyClientTrustFile.jksCCDcom.ibm.ssl.trustStore=
    \${install_root}/etc/DummyClientTrustFile.jks</argument> 9
</application-desc>
</jnlp>
```

- **1**--Specifies the mime type of the file must be JNLP so that the browser will know what to do with the file.
- **2**--Specifies that the application is depending on the `WASclient6.0` Java Runtime Environment and specifies the URL of the JNLP for the Application Clients run-time dependency.
- **3**--Specifies the JAR file containing the launcher class. This should be the first jar specified and must contain the URL of the JAR file.
- **4**--Specifies the EAR file to be downloaded, which is similar to the one you run on an Application Client for WebSphere Application Server installation.
- **5**--Specifies the value of the EAR file name of the J2EE application.
- **6**--Specifies an application descriptor.
- **7**--Specifies the arguments for the J2EE Application client application as they are specified on the launchClient call.
- **8, 9**--Overrides the values in the `sas.client.props` file. They are needed because the installation location of the Application Clients run-time dependency component is unknown before it is actually installed. By default, security is turned on for the client application, and these values are required. The ${*install_root*} directory name is substituted with the Application Clients run-time dependency component installation location at run time.

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. It is packaged in the `WebSphereClientLauncher.jar` file that is located in a WebSphere Application Server clients installation under the <*install_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, they 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.

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 client application. This JAR file is a JAR file that is already defined as a resource for the client 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 client application.

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\WASClient6.0_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\WASClient6.0_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
                 WASClient6.0_windows.jar myKeyAliasName

   jarsigner -keystore myKeystore -storepass myPassword
                 WebSphereClientRuntimeInstaller.jar myKeyAliasName
   ```

   a. This step also requires you to , such as myKeystore.

   b. You must also for the myKeystore file. For more information, see the topic, ″.″

   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 is an Installer JNLP
     It will download two .jars:
     WebSphereClientRuntimeInstaller.jar - includes the installer utility
     WASClient6.0_<platform>.jar - the client runtime JRE image

  The installer will unzip the client runtime jar on the client machine, and register
     it with Java Web Start

  --%>
<%! private final String description=″WebSphere Client 6.0 Runtime JRE″;
    // The version here is (WAS based) JRE version - as to be managed on the client
    private final String JREversion=″WASclient6.0″;%>
<%
      // locally declared variable
      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 Web Start is installed on the client,
      // this header will induce the browser to drive the Web Start Client
      response.setContentType(″application/x-java-jnlp-file″);    1
      response.setHeader(″Cache-Control″, null);
      response.setHeader(″Set-Cookie″, null);
```

```
      response.setHeader("Vary", null);

      // 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", JREversion);     2
      else
        response.addHeader("x-java-jnlp-version-id", JREversion);


%>
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for <%=description %> -->
<jnlp
  spec="1.0+" <%--
     Automate the code base response --%>
     codebase="<%=jnlpCodeBase%>"
  href="<%=jnlpRefURL%>">
 <information>
  <title><%=description%></title>
  <vendor>IBM</vendor>
  <icon href="icon.gif">
  <description><%=description%></description>
  <description kind="short"><%=description></description>
  <description kind="tooltip"><%=description></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="<%=JREversion%>"/> 5

 </resources>
 <resources os="Windows"> 6
   <jar href="windows/WASClient6.0_windows.jar"/> 7

   <%-- relative path of the jre executable --%>
   <property name="com.ibm.websphere.client.jre.launch.java"
 value="java\jre\bin\java.exe"/> 8
 <resources os="Linux">
  <jar href="linux/WASClient6.0_linux.jar"/>

  <property name="com.ibm.websphere.client.jre.launch.java" value="java/jre/bin/java"/>
 </resources>
 <installer-desc />
</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: 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 two configuration files, `sas.client.props` and `soap.client.props`.

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.
      type             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 *<install_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/WASClient6.0_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="WASclient6.0"/>
```

Preparing Application Clients run-time library component for Java Web Start:

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 directory to the installation bin directory. See the following example for help:

   ```
   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 , such as `myKeystore`.

   b. You must also for the `myKeystore` file. For more information, see the topic, ″.″
4. Create an Application Clients 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.
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 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
-->
<%! private final String description="WebSphere Jars";
%>
<%  // locally declared variable

String urlSt = request.getRequestURL().toString();
String jnlpCodeBase=urlSt.substring(0,urlSt.lastIndexOf('/'));
String jnlpRefURL=urlSt.substring(urlSt.lastIndexOf('/')+1,urlSt.length());
// The client application descriptor noted a resource reference to be resolved at deploy time as following
%>
<%--
 Need to set a JNLP mime type - if Web Start is installed on the client,
 this header will induce the browser to drive the Web Start 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);   2
%>
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for <%=description %> -->
<jnlp
  spec="1.0+"
 <%-- Automate the code base response
--%> codebase="<%=jnlpCodeBase%>"
href="<%=jnlpRefURL%>">
 <information>
  <title><%=description %></title>
  <description kind="short"><%=description %></description>
  <description kind="tooltip"><%=description %></description>
  <offline-allowed></offline-allowed>
 </information>
 <security>
 <all-permissions></all-permissions>
</security>
 <resources>
     <jar href="activation-impl.jar"/>   3
    <jar href="activity.jar"/>
    <jar href="activityImpl.jar"/>
    <jar href="activitySession.jar"/>
    <jar href="activitySessionPrivate.jar"/>
    <jar href="acwa.jar"/>
    <jar href="admin.jar"/>
    <jar href="annotations-core.jar"/>
    <jar href="appprofile-impl.jar"/>
    <jar href="appprofile.jar"/>
    <jar href="b2bjaxp.jar"/>

   <!-- ========================================  -->
    <!--                                          -->
    <!-- specify all the signed jars created by   -->
    <!-- buildClientlibJars tool                  -->
    <!--                                          -->
    <!-- ========================================  -->

    <jar href="wsif-j2c.jar"/>
    <jar href="wsif.jar"/>
    <jar href="wssec.jar"/>
    <jar href="wtp-util.jar"/>
    <jar href="wtpemf.jar"/>
    <jar href="xsd.jar"/>
    <jar href="xsd.resources.jar"/>
    <jar href="xss4j-dsig.jar"/>
    <jar href="xss4j-enc.jar"/>
 </resources>
 <component-desc/>
</jnlp>
```

- **1**--Specifies that the file is a JNLP mime type so that the browser can process the JNLP file.
- **2**--Specifies the Last-Modified header so that any changes to this JSP file are downloaded to the JNLP client.
- **3**--Specifies all the JAR files in the Application Clients run-time library component that are generated by the buildClientLibJars tool.

buildClientLibJars tool: 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\WASClient6.0_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
                   WASClient6.0_windows.jar myKeyAliasName

   jarsigner -keystore myKeystore -storepass myPassword
                   WebSphereClientRuntimeInstaller.jar myKeyAliasName
   ```

   a. This step also requires you to , such as myKeystore.

   b. You must also for the myKeystore file. For more information, see the topic, ″.″

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 , such as `myKeystore`.

   b. You must also for the `myKeystore` file. For more information, see the topic, ″.″

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
│   │          jnlp.jsp
│   │          WebSphereClientRuntimeInstaller.jar
│   ├──windows │
│   │               WASClient6.0_windows.jar
│   │
│   └──WebSphereJars
│                     jnlp.jsp
│                     activities.jar
│                          :
│                     (all the jars created in step 7 under
│                       c:\WebApp1\Runtime\WebSphereJars)
│
├──theme
│              Master.css
│
└──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).

Deploying application clients on z/OS

For J2EE application clients that will run on z/OS or OS/390, you may use one of the following options to define resources:

- Run the Application Client Resource Configuration Tool (ACRCT) on Windows.
- Run the Application Client Resource Configuration Tool (ACRCT) scripting tool on z/OS.

Both options produce identical output with one possible exception: the sequence in which resource definitions are stored in the Enterprise Archive (EAR) file of the application client. The client container on z/OS uses these resource definitions for resolving and creating an instance of the resources for the application client.

Before you begin: Make sure you have completed the following tasks:

1. Develop the J2EE application client according to guidelines.
2. Assemble the application client. See ″Assembling application clients″ in the information center.
3. Find out what resources are available on the z/OS system on which you will install the client. These resources include:
   - Enterprise beans
   - JMS message resources
   - JDBC databases

- Java Mail providers
- Environment entries (native types)
- URLs

4. Decide whether you want to provide resource properties for the ACRCT scripting tool on the command line or through an input file. If you do not specify required properties, the ACRCT scripting tool issues an error message to the MVS console and ends its processing.

   Recommendation: Determine which resource or provider properties are required.

1. Use the administrative console to install the application client on z/OS or OS/390.
2. (Optional) Set up a plain text input file to provide on the command line when you start the ACRCT scripting tool.

   Rules:
   - Each line in the input file may contain only one key and value pair that defines a property of the resource to be configured.
   - For each resource to be configured, determine which resource or provider properties are required.
   - Follow the syntax rules explained in Application Client Resource Configuration Tool (ACRCT) Scripting tool for z/OS.
   - You may define your own properties for the resource, using the format property.name=value.

   Sample: Input file for a data source provider:

   ```
   providertype=DataSourceProvider
   providername=DB2UDBV7
   name=″PolicyDatasource″
   description=″Datasource for Policy App″
   jndiname=jdbc/PolicyDS
   databasename=POLICYAPP
   user=dbuser
   password=dbpw
   reenterpassword=dbpw
   property.my.resource.property.one=value1
   property.my.resource.property.two=value2
   ```

3. On z/OS or OS/390, start the ACRCT scripting tool by invoking the shell script clientConfig in the UNIX System Services (USS) environment.

   Example:

   ```
   /usr/lpp/WebSphere/V5R0M0/bin/clientConfig.sh
   ```

   Rule: You must specify the application client's Enterprise Archive (EAR) file on the command line. You may either specify resource parameters directly on the command line, or specify an input file. The syntax and parameter descriptions appear in Application Client Resource Configuration Tool (ACRCT) Scripting tool for z/OS.
   - If the resource parameters are properly specified, the ACRCT scripting tool updates the application client's client-resources.xmi file with appropriate resource definitions.
   - If the resource parameters are not properly specified or are missing, the ACRCT scripting tool issues an error message to the MVS console, and ends its processing.

   Tip: If you receive an error message in response to the invocation, consider using the help function described in Application Client Resource Configuration Tool (ACRCT) Scripting tool for z/OS.

When the scripting tool successfully completes, the application client's EAR file is updated with the appropriate resource definitions.

When you have finished defining or updating the application client's resources, launch the application client.

Installing the Client Development Kit for z/OS

The Client Development Kit (CDK) for z/OS contains the following command-line tools that are used to configure and deploy client applications for WebSphere Application Server:

- **clientConfig**

  The **clientConfig** command is used to start the Application Client Resource Configuration Tool (ACRCT). The most common task that you perform with the ACRCT is opening and modifying the components of enterprise archive (EAR) files.
- **clientUpgrade**

  The **clientUpgrade** command migrates the V4.0.x and V5.x client resources to the V6.0 level resources.
- **endptEnabler**

  The **endptEnabler** command enables a set of Web services within an EAR file. See ″endptEnabler command″ in the information center for additional information.
- **Java2WSDL**

  The **Java2WSDL** command maps a Java class to a Web Services Description Language (WSDL) file by following the Java API for XML-based Remote Procedure Call (JAX-RPC) specification. See Java2WSDL command″ in the information center for additional information.
- **WSDL2Java**

  The **WSDL2Java** command is run against the WSDL file to create Java APIs and deployment descriptor templates according to the JAX-RPC and Web Services for Java 2 Platform, Enterprise Edition (J2EE) specifications. See ″WSDL2Java command″ in the information center for additional information.
- **wsdeploy**

  The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application EAR file or an application client Java archive (JAR) file.
- **EJBDeploy**

  The EJB deployment tool provides a command-line interface (ejbdeploy) that you can use to generate enterprise bean deployment code. To learn more about this command line refer to the Application Server Toolkit information center.
- **UDDI Utility Tools**

  The UDDI Utility Tools is a suite of functions that can be used to migrate, move, and copy UDDI Version 2 entities, including child entities and their respective Version 2 entity keys, into a Version 3 UDDI Registry.

The Client Development Kit is included with the WebSphere Application Server z/OS product and located in the following directory:

```
$WAS_HOME/downloads/ClientDevKit/setup.exe
```

When the Client Development Kit is installed, the directory looks as if it is a Windows platform directory.

To install the Client Development Kit:

1. Install and Customize your z/OS WebSphere Application Server. See ″Task overview: installing″ in the information center.
2. Locate the Client Development Kit executable file in the `$WAS_HOME/downloads/ClientDevKit/` directory. FTP the **setup.exe** file in binary mode to your Windows-based client machine.
3. Execute the **setup.exe** installation file.
4. Follow the InstallWizard prompts to complete the installation.

You are now ready to configure applications that can be deployed into the WebSphere Application server runtime.

Application Client Resource Configuration Scripting tool for z/OS

This section describes the command line syntax for the z/OS scripting version of the Application Client Resource Configuration Tool (ACRCT). The ACRCT scripting tool for z/OS allows you to:
- Define or delete resources for an application client that will run on z/OS.
- List the properties of a resource or provider that is already defined for the application client.

For define and delete actions, the ACRCT scripting tool alters the Enterprise Archive (EAR) file for the application client, as instructed by options you specify. If the ACRCT scripting tool encounters an error at any time during its processing, the tool issues an error message to the MVS console and terminates without changing the original contents of the application client EAR file.

When you use the ACRCT scripting tool, you may specify:
- More than one action (define, delete, or list) to perform for a specific application client.
- More than one EAR file, to define or delete resources for more than one application client at a time.

The command line invocation syntax for the ACRCT scripting tool follows. When you have a choice of one required keyword, those keywords appear within brackets [].
- To define or delete resources:

```
acrct -earfile earfile [-define | -delete] [-provider | -resource]
[-f inputfile | key=value]
```
- To list resources:

```
acrct -list [-provider | -resource] [-f inputfile | -p key=value]
```
- To get help information:

```
acrct -help
```

Parameters

where:

-earfile
Is a required parameter that indicates the input filename of the application client EAR file.

earfilename
Identifies the location and name of the EAR file that contains the application client. This path and filename must directly follow the -earfile parameter.

-define
Instructs the scripting tool to define a provider or resource based on the input properties.

-delete
Instructs the scripting tool to delete a provider or resource based on the input properties.

-list
Instructs the scripting tool to the properties of a particular provider or resource, based on the input properties.

-help
Instructs the scripting tool to list basic examples and guidelines for using quotes around key values.

-provider
Indicates that the object to be defined or deleted, or for which properties are to be listed, is a provider.

-resource
Indicates that the object to be defined or deleted, or for which properties are to be listed, is a resource.

-f Indicates that the input properties for the provider or resource are provided in an input file, rather than specified directly on the command line.

inputfile
Identifies the location and name of the input file that contains the provider or resource properties. This path and filename must directly follow the -f parameter.

key=*value*
Specifies an input property for the provider or resource, in the form of key and value pairs.

Rules:
- You must use lowercase for keys.
- You cannot use blanks within a key and value pair; a blank signals the end of one key/value pair.
- Because blanks separate key and value pairs, you must be careful when a value you supply contains blanks. When you specify a value that contains blanks, enclose the value in single quotes or double quotes. Because some shells processes quotes differently, you might have to do some testing to determine whether you must use single or double quotes.

  Example: Suppose you invoke the scripting tool, passing this input:

  ```
  /WebSphere/V5R0M0/AppServer/bin:>acrct  -earfile
  usr/lpp/myapps/applclient2.ear  -define
  -provider  providername='WebSphere  JMS  Provider'
  ```

  The response is an error message along with an echo of the input string that the shell receives, followed by the input string as the scripting tool will process it:

  ```
  -earfile  usr/lpp/myapps/applclient2.ear  -define
  -provider  providername=WebSphere  JMS  Provider


  String  to  be  parsed  by  the  Scripting  Tool:

  -earfile  usr/lpp/myapps/applclient2.ear  -define
  -provider  providername=WebSphere  JMS  Provider


  Ear  file  is  missing  or  is  improperly  specified.
  Invalid  syntax:  -earfile  usr/lpp/myapps/applclient2.ear  -define  -provider  providername=
  WebSphere  JMS  Provider
  ```

  As you can see from the response, the shell has stripped off the single quotes, and passes invalid input to the scripting tool. To correct the problem, you need to use double quotes.
- The number of key and value pairs you specify depends on the type of resource or provider you are configuring. For each resource to be configured, use this information to determine which resource or provider properties are required.

The following examples demonstrate correct syntax:

Defining a new provider for an application client, using an input file:
```
acrct -earfile usr/lpp/myapps/applclient1.ear -define -provider -f
usr/lpp/myapps/inputProvider1.def
```
Defining a new provider for an application client, specifying properties directly on the command line:
```
acrct -earfile usr/lpp/myapps/applclient2.ear -define -provider -p
providertype=DataSourceProvider name=DB2UDBV7
```
Defining a new provider and deleting the resource it replaces, in the same EAR file:
```
acrct -earfile usr/lpp/myapps/applclient1.ear -define -provider -f
usr/lpp/myapps/inputProvider2.def -delete -resource -f usr/lpp/myapps/inputProvider1.def
```
Defining a new provider in more than one EAR file:
```
acrct -earfile usr/lpp/myapps/applclient1.ear -define -provider -f
usr/lpp/myapps/inputProvider2.def -earfile usr/lpp/myapps/applclient2.ear -define
-provider -f usr/lpp/myapps/inputProvider2.def
```

Determining required properties for z/OS application client resources

When you deploy application clients on z/OS or OS/390, you need to determine the required properties to specify when using the z/OS scripting version of the Application Client Resource Configuration Tool (ACRCT).

Note: This procedure applies only for J2EE application clients.

Use the following information to determine required properties to specify for application client resources.

If the application uses this type of resource or provider:	Find required and optional properties in these articles:
Data source	• Datasource Provider • Datasource
JMS	• JMS Provider • JMS Connection • JMS Destination
Mail session	• Mail provider • Mail session
Resource environment	• Resource environment provider • Resource environment entry
URL	• URL provider • URL factory
WebSphere MQ queue	• WebSphere MQ queue connection factory • WebSphere MQ queue destination factory
WebSphere MQ topic	• WebSphere MQ topic connection factory • WebSphere MQ topic destination factory
WebSphere queue	• WebSphere queue connection factory • WebSphere queue destination factory
WebSphere topic	• WebSphere topic connection factory • WebSphere topic destination factory

Properties for data source providers:

Type	Value	Description
name	required	
description	optional	
implementation	required	
classpath	required	
native library path	required for specific providers	required for • DB2 for zOS JDBC Provider (RRS) • DB2 Universal JDBC Driver Provider (optional for all others)

Properties for data sources:

Type	Value
providertype=DataSourceProvider	required
providername	required

Type	Value
name	required
description	optional
jndiname	required
databasename	optional
user	optional
password	optional

providertype=DataSourceProvider
providername=DB2UDBV7
name=″PolicyDatasource″
description=″Datasource for Policy App″
jndiname=jdbc/PolicyDS
databasename=POLICYAPP
user=dbuser
password=dbpw

Properties for JMS providers:

Type	Value
name	required
description	optional
classpath	optional
externalinitialcontextfactory	optional
externalproviderurl	optional

Properties for JMS connections:

Type	Value	Description
providertype=JMSProvider	required	
providername	required	
name	required	
description	optional	
jndiname	required	
externaljndiname	optional	
type	required	Valid values are: • **QUEUE** • **TOPIC**
user	optional	
password	optional	

Properties for JMS destinations:

Type	Value	Description
providertype=JMSProvider	required	
providername	required	

Type	Value	Description
name	required	
description	optional	
jndiname	required	
externaljndiname	optional	
type	required	Valid values are: • **QUEUE** • **TOPIC**

Properties for mail providers:

Type	Value
name	required
description	optional
classpath	optional

Properties for mail sessions:

Type	Value	Description
providertype=MailProvider	required	
providername	required	
name	required	
description	optional	
jndiname	required	
mailfrom	optional	
mailstorehost	optional	
mailstoreuser	optional	
mailstorepassword	optional	
mailtransporthost	optional	
mailtransportprotocol	required	Valid values are: • **smtp** • **imap** • **pop3**
mailtransportuser	optional	
mailtransportpassword	optional	
debug	required	Valid values are **True** or **False**.
strict	required	Valid values are **True** or **False**.

Properties for resource environment providers:

Type	Value
name	required
description	optional
classpath	optional

Properties for resource environment entries:

Type	Value
providertype=ResourceEnvironmentProvider	required
providername	required
name	required
description	optional
jndiname	required

Properties for URL providers:

Type	Value
name	required
description	optional
protocol	optional
classpath	optional
streamhandlerclass	optional

Properties for URL factories:

Type	Value
providertype=URLProvider	required
providername	required
name	required
description	optional
jndiname	required
url	required

Properties for WebSphere MQ queue connection factories:

Type	Value	Description
providertype=JMSProvider	required	
providername=″MQ JMS Provider″	required	
name	required	
description	optional	
jndiname	required	
transporttype	required	Valid values are: • **CLIENT** • **BINDINGS**
clientid	optional	
user	optional	
password	optional	
channel	optional	
ccsid	optional	

Properties for WebSphere MQ queue destination factories:

Type	Value	Description
providertype=JMSProvider	required	
providername=″MQ JMS Provider″	required	
name	required	
description	optional	
jndiname	required	
persistence	required	Valid values are: • **APPLICATION_DEFINED** • **QUEUE_DEFINED** • **PERSISTENT** • **NONPERSISTENT**
priority	required	Valid values are: • **APPLICATION_DEFINED** • **QUEUE_DEFINED** • ***specified_integer*** **from 0 through 9**
expiry	required	Valid values are: • **APPLICATION_DEFINED** • **UNLIMITED** • ***specified_value***
basequeuename	required	
basequeuemanagername	optional	
targetclient	required	Valid values are: • **JMS** • **MQ**
ccsid	optional	
usenativeencoding	required	Valid values are: • **True** • **False**

Properties for WebSphere MQ topic connection factories:

Type	Value	Description
providertype=JMSProvider	required	
providername=″MQ JMS Provider″	required	
name	required	
description	optional	
jndiname	required	
transporttype	required	Valid values are: • **CLIENT** • **BINDINGS**
clientid	optional	
brokercontrolqueue	optional	
brokerqueuemanager	optional	
brokerpubqueue	optional	
brokersubqueue	optional	

Type	Value	Description
brokerccsubq	optional	
brokerversion	required	Valid values are: • **MA0C** • **MQSI**
userid	optional	
password	optional	
ccsid	optional	
channel	optional	

Properties for WebSphere MQ topic destination factories:

Type	Value	Description
providertype=JMSProvider	required	
providername=″MQ JMS Provider″	required	
name	required	
description	optional	
jndiname	required	
persistence	required	Valid values are: • **APPLICATION_DEFINED** • **QUEUE_DEFINED** • **PERSISTENT** • **NONPERSISTENT**
priority	required	Valid values are: • **APPLICATION_DEFINED** • **QUEUE_DEFINED** • ***specified_integer*** from 0 through 9
expiry	required	Valid values are: • **APPLICATION_DEFINED** • **UNLIMITED** • ***specified_value***
basetopicname	required	
targetclient	required	Valid values are: • **JMS** • **MQ**
brokerdursubqueue	optional	
brokerccdursubqueue	optional	
ccsid	optional	
usenativeencoding	required	Valid values are: • **True** • **False**

Properties for WebSphere queue connection factories:

Type	Value
providertype=JMSProvider	required
providername=″WebSphere JMS Provider″	required

Type	Value
name	required
description	optional
jndiname	required
node	required
servername	required
user	optional
password	optional

Properties for WebSphere queue destination factories:

Type	Value	Description
providertype=JMSProvider	required	
providername=″WebSphere JMS Provider″	required	
name	required	
description	optional	
jndiname	required	
node	required	
persistence	required	Valid values are: • **APPLICATION_DEFINED** • **PERSISTENT** • **NONPERSISTENT**
priority	required	Valid values are: • **APPLICATION_DEFINED** • ***specified_integer*** from 0 through 9
expiry	required	Valid values are: • **APPLICATION_DEFINED** • **UNLIMITED** • ***specified_value***

Properties for WebSphere topic connection factories:

Type	Value	Description
providertype=JMSProvider	required	
providername=″WebSphere JMS Provider″	required	
name	required	
description	optional	
jndiname	required	
servername	required	
node	required	
port	required	Valid values are: • **QUEUED** • **DIRECT**
clientid	optional	

Type	Value	Description
userid	optional	
password	optional	

Properties for WebSphere topic destination factories:

Type	Value	Description
providertype=JMSProvider	required	
providername=″WebSphere JMS Provider″	required	
name	required	
description	optional	
jndiname	required	
topic	required	
persistence	required	Valid values are: • **APPLICATION_DEFINED** • **PERSISTENT** • **NONPERSISTENT**
priority	required	Valid values are: • **APPLICATION_DEFINED** • ***specified_integer*** from 0 through 9
expiry	required	Valid values are: • **APPLICATION_DEFINED** • **UNLIMITED** • ***specified_value***

Deploying J2EE application clients on workstation platforms

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.bat command (on Windows systems) or setupCmdLine.sh (on UNIX platforms) command 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

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.

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}

clientRAR tool:

This section 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:

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

Starting the Application Client Resource Configuration Tool and 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 `install_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 might 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:

Configuring new data source providers (JDBC providers) for application clients

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: 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.

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 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

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** > **JavaMail Providers** > ***java_mail_provider_instance*** > **JavaMail 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: 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

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 **JavaMail Providers** > **MailProvider** > **JavaMail Sessions**. Complete one of the following actions:
   - Right click the **JavaMail Sessions** folder and select **New Factory**.
   - Click **Edit** > **New** on the menu bar.
5. Configure the JavaMail 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

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 Provider**.
   - 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:

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: 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

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. Do 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 Factory**.
   - 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.

WebSphere asynchronous messaging using the Java Message Service API for the Application Client Resource Configuration Tool

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 an overview of asynchronous messaging using JMS support provided by the 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.

Configuring Java messaging client resources

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.

During this task, 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*.

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. Right-click **Messaging Providers** and select **New**.
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:

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.

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.

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

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: 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 **JMS Connection Factories** folder.
3. Click the connection factory folder, and complete one of the following actions:
   - Right-click the folder and select**New Factory**.
   - 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 **JMS 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.

Example: Configuring MQ Queue and Topic connection factories and destination factories for application clients

The purpose of this article is to help you configure MQ Queue connection factory, MQ Topic connection factory, MQ Queue destination factory, and MQ Topic destination factory settings.

- Required fields:
  - MQ Queue Connection Factory Properties page: name, jndiName and transport type
  - MQ Topic Connection Factory Properties page: name, jndiName and broker Version
  - MQ Queue Factory Properties page: name, jndiName, persistence, priority, expiry, baseQueueName and targetClient
  - MQ Topic Factory Properties page: name, jndiName, persistence, priority, expiry, baseQueueName and targetClient
- Special cases:
  - The transport type must be `CLIENT`, or `BINDINGS`.

- The Broker Version must be MA0C, or MQSI.
- The port must be a numerical value between -2417483648 and 2417483647.
- The CCSID must be a numerical value between -2417483648 and 2417483647.
- The persistence value must be APPLICATION_DEFINED, QUEUE_DEFINED, PERSISTENT or, NONPERSISTENT.
- The priority must be APPLICATION_DEFINED, QUEUE_DEFINED, or SPECIFIED.
- The expiry must be APPLICATION_DEFINED, UNLIMITED, or SPECIFIED.
- The integer encoding must be Normal, or Reversed.
- The decimal encoding must be Normal, or Reversed.
- The floating encoding must be IEEENormal, IEEEReversed or S390.
- The target client must be JMS or MQ.
- On the MQ Queue Connection Factory Properties page, only set the queueManager, host, and port values. These are required fields if the transport type is CLIENT.
- On the MQ Topic Connection Factory Properties page, only set the queueManager, host, and port (required) fields if the transport type is CLIENT.
- On the MQ Topic Factory Properties, and the MQ Queue Factory Properties pages, only set the Integer encoding, decimal encoding, and floating point encoding (required) fields if you do not set the nativeEncoding value.
- On the MQ Topic Factory Properties and the MQ Queue Factory Properties pages, the specified priority entry field must be an integer between 0 and 9 if priority is set to SPECIFIED .
- On the MQ Topic Factory Properties and the MQ Queue Factory Properties pages, the specified expiry entry field must be a value greater than 0 if the expiry value is set to SPECIFIED.

• Example:

```
<resources.jms:JMSProvider xmi:id="JMSProvider_1" name="MQ JMS Provider"
description="mqJMSProvider:description"
externalInitialContextFactory="mqJMSProvider:contextFactoryClass"
externalProviderURL="mqJMSProvider:providerUrl">
<classpath>mqJMSProvider:classpath</classpath>
<factories xmi:type="resources.jms.mqseries:MQQueueConnectionFactory"
xmi:id="MQQueueConnectionFactory_1" name="mqQCF:name" jndiName="mqQCF:jndiName"
description="mqQCF:description" userID="mqQCF:user" password="{xor}Mi4OHBllMT4yOg=="
queueManager="mqQCF:queueManager" host="mqQCF:host" port="1" channel="mqQCF:channel"
transportType="CLIENT" clientID="mqQCF:clientId" CCSID="2">
<propertySet xmi:id="J2EEResourcePropertySet_3">
<resourceProperties xmi:id="J2EEResourceProperty_3" name="mqQCF:customName"
value="mqQCF:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.mqseries:MQTopicConnectionFactory"
xmi:id="MQTopicConnectionFactory_1" name="mqTCF:name" jndiName="mqTCF:jndiName"
description="mqTCF:description" userID="mqTCF:user"
password="{xor}Mi4LHBllNTE7NhE+Mjo=" host="mqTCF:host" port="1"
transportType="CLIENT" channel="mqTCF:channel" queueManager="mqTCF:queueManager"
brokerControlQueue="mqTCF:brokerControlQueue"
brokerQueueManager="mqTCF:brokerQueueManager" brokerPubQueue="mqTCF:brokerPubQueue"
brokerSubQueue="mqTCF:brokerSubQueue" brokerCCSubQ="mqTCF:brokerCCSubQ"
brokerVersion="MA0C" clientID="mqTCF:clientId" CCSID="2">
<propertySet xmi:id="J2EEResourcePropertySet_4">
<resourceProperties xmi:id="J2EEResourceProperty_4" name="mqTCF:customName"
value="mqTCF:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.mqseries:MQQueue" xmi:id="MQQueue_1" name="mqQ:name"
jndiName="mqQ:jndiName" description="mqQ:description" persistence="APPLICATION_DEFINED"
priority="SPECIFIED" specifiedPriority="1" expiry="SPECIFIED" specifiedExpiry="1"
baseQueueName="mqQ:baseQueueName" baseQueueManagerName="mqQ:baseQueueManagerName"
CCSID="1" integerEncoding="Normal" decimalEncoding="Normal"
floatingPointEncoding="IEEENormal" targetClient="JMS">
<propertySet xmi:id="J2EEResourcePropertySet_5">
<resourceProperties xmi:id="J2EEResourceProperty_5" name="mqQ:customName"
value="mqQ:customValue"/>
</propertySet>
</factories>
```

```
<factories xmi:type="resources.jms.mqseries:MQTopic" xmi:id="MQTopic_1"
name="mqT:name" jndiName="mqT:jndiName" description="mqT:description"
persistence="APPLICATION_DEFINED" priority="SPECIFIED" specifiedPriority="1"
expiry="SPECIFIED" specifiedExpiry="2" baseTopicName="mqT:baseTopicName" CCSID="3"
integerEncoding="Normal" decimalEncoding="Normal" floatingPointEncoding="IEEENormal"
targetClient="JMS" brokerDurSubQueue="mqT:brokerDurSubQueue"
brokerCCDurSubQueue="mqT:brokerCCDurSubQueue">
<propertySet xmi:id="J2EEResourcePropertySet_6">
<resourceProperties xmi:id="J2EEResourceProperty_6" name="mqT:customName"
value="mqT:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_7">
<resourceProperties xmi:id="J2EEResourceProperty_7" name="mqJMSProvider:customName"
value="mqJMSProvider:customValue"/>
</propertySet>
</resources.jms:JMSProvider>
```

Example: Configuring WAS Queue and Topic connection factories and destination factories for application clients

The purpose of this article is to help you configure Queue connection factory, Topic connection factory, Queue destination factory, and Topic destination factory settings.

- Required fields include:
  - Java Message Service (JMS) Provider Properties page: name
  - WebSphere Application Server Queue Connection Factory Properties page: name, jndiName and node
  - WebSphere Application Server Topic Connection Factory Properties page: name, jndiName, node and port
  - WebSphere Application Server Queue Factory Properties page: name, jndiName, node, persistence, priority and expiry
  - WebSphere Application Server Topic Factory Properties page: name, jndiName, topic name, persistence, priority and expiry
- Special cases:
  - The port value must be `QUEUED` or `DIRECT`.
  - The CCSID must be a numerical value between -2417483648 and 2417483647.
  - The persistence value must be `APPLICATION_DEFINED`, `PERSISTENT`, or `NONPERSISTENT`.
  - The priority value must be `APPLICATION_DEFINED`, or `SPECIFIED`.
  - The expiry value must be `APPLICATION_DEFINED`, `UNLIMITED`, or `SPECIFIED`.
  - On the WAS Topic Factory Properties, and the WAS Queue Factory Properties pages, the specified priority entry field must be an integer between 0 and 9, if the priority value is set to `SPECIFIED` .
  - On the WAS Topic Factory Properties, and the WAS Queue Factory Properties pages, the specified expiry entry field must be a value greater than 0 if expiry is set to `SPECIFIED`.
- Example:

```
<resources.jms:JMSProvider xmi:id="JMSProvider_2" name="WebSphere JMS Provider"
description="wasJMSProvider:description"
externalInitialContextFactory="wasJMSProvider:contextfactoryclass"
externalProviderURL="wasJMSProvider:providerURL">
<classpath>wasJMSProvider:classpath</classpath>
<factories xmi:type="resources.jms.internalmessaging:WASQueueConnectionFactory"
xmi:id="WASQueueConnectionFactory_1" name="wasQCF:name" jndiName="wasQCF:jndiName"
description="wasQCF:description" userID="wasQCF:user" password="{xor}KD4sDhwZZSosOi0="
node="wasQCF:Node">
<propertySet xmi:id="J2EEResourcePropertySet_8">
<resourceProperties xmi:id="J2EEResourceProperty_8" name="wasQCF:customName"
value="wasQCF:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.internalmessaging:WASTopicConnectionFactory"
xmi:id="WASTopicConnectionFactory_1" name="wasTCF:name" jndiName="wasTCF:jndiName"
description="wasTCF:description" userID="wasTCF:user" password="{xor}KD4sCxwZZTE+Mjo="
node="wasTCF:node" port="QUEUED" clientID="wasTCF:clientId">
<propertySet xmi:id="J2EEResourcePropertySet_9">
```

```
<resourceProperties xmi:id="J2EEResourceProperty_9" name="wasTCF:customName"
value="wasTCF:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.internalmessaging:WASQueue" xmi:id="WASQueue_1"
name="wasQ:name" jndiName="wasQ:jndiName" description="wasQ:description"
node="wasQ:node" persistence="APPLICATION_DEFINED" priority="SPECIFIED"
specifiedPriority="1" expiry="SPECIFIED" specifiedExpiry="1">
<propertySet xmi:id="J2EEResourcePropertySet_10">
<resourceProperties xmi:id="J2EEResourceProperty_10" name="wasQ:customName"
value="wasQ:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms.internalmessaging:WASTopic" xmi:id="WASTopic_1"
name="wasT:name" jndiName="wasT:jndiName" description="wasT:description"
topic="wasT:topicName" persistence="APPLICATION_DEFINED" priority="SPECIFIED"
specifiedPriority="1" expiry="SPECIFIED" specifiedExpiry="1">
<propertySet xmi:id="J2EEResourcePropertySet_11">
<resourceProperties xmi:id="J2EEResourceProperty_11" name="wasT:customName"
value="wasT:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_12">
<resourceProperties xmi:id="J2EEResourceProperty_12" name="wasJMSProvider:customName"
value="wasJMSProvider:customValue"/>
</propertySet>
</resources.jms:JMSProvider>
```

Configuring new resource environment providers for application clients

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 Provider**.
   - 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

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 entry**, 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.

Creating locally defined objects for message destination references and message destinations

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 application client `.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.

When a local object definition is created using the ACRCT, the JNDI name of the local object definition points to the reference for which the local object definition applies. The object might be a `resource-ref`, `resource-env-ref` or `message-destination-ref`. If the `message-destination-ref` has a `message-destination-link`, then point the local object definition to the message-destination. Local object definitions either point to the `message-destination-ref` (if the `message-destination-ref` has no link) or to the `message-destination` (if the message-destination-ref has a link). Any local object definitions pointing to `message-destination-refs` that have a link are ignored.

1. Start an assembly tool such as Application Server Toolkit (AST) or Rational Web Developer, and open an EAR file.
2. Create a locally defined object.
3. Point the object to the appropriate message destination.
4. Save the EAR file.

Managing application clients

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:

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
   - Data source 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:

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:

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:

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:

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:

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: 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: 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:

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**.

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.

Using Web services makes most sense if your application clients are non-J2EE applications, unless you have J2EE applications spread across the Web. It is recommended that you use J2EE technologies if all your clients are J2EE applications because performance can decrease when you use a Web service in a J2EE exclusive environment.

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 Web service client. Now, the accounting application can 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.

Use Web services applications with WebSphere Application Server by following the steps provided:

1. Plan to use Web services. Review the Universal Description, Discovery, and Integration (UDDI) and Web services enablement of the service integration bus (SIBWS) concepts to learn how these components can make your Web services plan more robust.
2. (Optional) Migrate existing Web services. See ″Migrating Apache SOAP Web services to Web Services for J2EE standards″ in the information center.

   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. See ″Developing Web services applications″.

   This topic is a good starting point in learning about how to develop a J2EE Web service.
4. Configure Web services deployment descriptors. See ″Configuring Web services deployment descriptors″ in the information center.

   You need to configure the deployment descriptors so that WebSphere Application Server can process the incoming Web services requests.
5. Assemble Web services. See ″Assembling Web services applications″.

   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 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. See ″Developing 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. I If you are using V5.x, refer to ″Securing Web services for version 5.x applications based on WS-Security″in the information center. If you are using V6.x, refer to ″Securing Web services for version 6 applications based on WS-Security″
11. Tune Web services. See ″Monitoring the performance of Web services applications″ in the information center.

    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. See ″Troubleshooting 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. One way to find new suppliers is to use a Universal Description, Discovery and Integration (UDDI) registry to search for potential suppliers. When the suppliers are chosen, the registry sends back information on how to contact the flower distributors that meet the flower shop owner's criteria.

The flower shop owner can request price lists from each of the suppliers by obtaining a Web Services Description Language (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 ″Overview: Online garden retailer Web services scenarios″ 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.
- Universal Description, Discovery and Integration Protocol (UDDI)

  UDDI is the registry that hosts the service broker. UDDI is similar to the Yellow Pages in a phone book.

For a more detailed scenario, see ″Web services scenario: Overview″ in the information center, 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.

Version 6.0 uses Web Services for J2EE 1.1 as the standard for developing and implementing Web services. Web Services for J2EE 1.1 is one of the Web service APIs available in J2EE 1.4.

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.

To review the entire Web Services for J2EE specification, see 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; most of these rules are from RMI with some additions for JAX-RPC. The following are the rules for defining a JAX-RPC interface:

- 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.

To learn more about JAX-RPC see Web services: Resources for learning.

SOAP

Simple Object Access Protocol (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.

For more information about SOAP, see Web services: Resources for learning.

SOAP with Attachments API for Java

SOAP with Attachments API for Java (SAAJ) is used for SOAP messaging that works behind the scenes in the Java API for XML-based RPC (JAX-RPC) implementation.

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. The SOAP message is manipulated using the SAAJ data model. The primary interface in the SAAJ model is javax.xml.soap.SOAPElement.

WebSphere Application Server uses SAAJ Version 1.2. The main benefit of SAAJ Version 1.2 is that the model extends the Document Object Model (DOM) model. The DOM model is used by applications that manipulate XML. Using this model applications are able to 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.

To review the entire SAAJ API, see 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. A link to this Web site is listed in Web services: Resources for learning.

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 change encoding from UTF-8 to UTF-16″ in the information center 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.

To learn more about the WS-Basic profile, including scenarios, UTF and BOM, see Web services: Resources for learning.

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.

Web services migration best practices

This topic presents best practices when migrating Web services applications.

Migrating a Version 5 JAX-RPC client that uses SOAP over JMS to invoke a Web service

A Java API for XML-based remote procedure call (JAX-RPC) client running on WebSphere Application Server Version 5, can use SOAP over Java Messaging Service to invoke a Web service that is running on a Version 5 application server.

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

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

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

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

- Disable global security on the Global security panel within the administrative console. To disable global security, click **Security > Global security**. Deselect the **Enable global security** option.
- Modify the settings for the service integration bus that hosts the queue destination so that the bus security is disabled and the bus does not inherit security characteristics from the server or the cell. This option is equivalent to the level of security that you can configure in Version 5.
- Configure the basic authentication on each client that uses the service. See ″Configuring HTTP basic authentication with the administrative console″ in the information center.

Migrating Apache SOAP Web services

See ″Migrating Apache SOAP Web services to Web Services for J2EE standards″ in the information center to learn how to migrate Apache SOAP Web services. This topic explains how to migrate Web services that were developed using Apache SOAP to Web services that are developed based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

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

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

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

.

Web services: Resources for learning

This topic provides relevant supplemental information about the following Web services-related topics:

- Web services overview
- Developing Web services:

  Includes developing Web services based on the Java 2 Platform, Enterprise Edition (J2EE) and Java API for XML-based remote procedure call (JAX-RPC) specifications.
- Performance

  Includes key Web sites that discuss performance best practices.
- Universal Description Discovery and Integration (UDDI)

  Includes an overview about UDDI and information about the UDDI Java API.
- The Web Services Invocation Framework (WSIF)

  Includes a look into the Apache Software Foundation and its maintenance of WSIF.
- Web Services-Interoperability (WS-I) Basic Profile

  Includes an overview about the WS-I Basic Profile.
- SOAP

  Includes an overview about SOAP, information about the SOAP syntax and processing rules.
- Security

  Includes a roadmap to security, the WS-Security specification, best practices, a profile of the OASIS Security Assertion Markup Language (SAML) and more.
- Samples

  Includes 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 5.1.1 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 Java 2 Platform, Enterprise Edition (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 some of the basic JAX-RPC programming concepts. It describes the JAX-RPC client and server programming models and provides some simple examples for illustration. 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 Universal Description, Discovery, and Integration (UDDI).
- 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 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. 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 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. Web Services security also provides a general-purpose mechanism for associating security tokens with messages. Additionally, Web Services Security 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 Web Services 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 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 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, 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 Web Services 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 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 Application Server.

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. See "Securing Web services for version 5.x applications based on WS-Security" in the information center.

wsdeploy command

The Client Development Kit (CDK) for z/OS contains the wsdeploy command-line tool needed for deploying Web services. See Installing the Client Development Kit to start using the tool.

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″ in the information center 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**. See ″Configuring a Web service client deployed WSDL file name with the wsadmin tool″ in the information center.

To configure client bindings through the administrative console:

1. Open the administrative console.

2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Web modules** > *module_instance* > **Web services client bindings**.

   For EJB modules, click **Applications** >**Enterprise Applications** > *application_instance* > **EJB 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.

To view this page, click **Applications** >**Enterprise Applications** > *application_instance* > **Web modules** > *module_instance*>**Web services client bindings**.

For EJB modules, click **Applications** >**Enterprise Applications** > *application_instance* > **EJB modules** > *module_instance*>**Web services client bindings**.

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* > **Web modules** > *module_instance*>**Web services client bindings** > **Edit**> *preferred_port_instance*.

For EJB modules, click **Applications** >**Enterprise Application** > *application_instance* > **EJB 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* > **Web modules** > *module_instance*>**Web services client bindings** > **Edit**.

For EJB modules, click **Applications** >**Enterprise Applications** > *application_instance* > **EJB modules** > *module_instance*>**Web services client bindings** > **Edit**.

Port:

Identifies the name of a port.

Request timeout:

Specifies the time, in seconds, that the 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.

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. See ″Configuring the scope of a Web service port with the wsadmin tool″ in the information center.

To change the scope setting through the administrative console:

1. Open the administrative console.
2. Click **Applications** >**Enterprise Applications** > *application_instance* > **Web Modules** > *module_instance*>**Web Services Implementation Scope**. If you are using an EJB module click **EJB Modules** instead of **Web Modules**.
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 determines when a new implementation instance is created for Web service ports. For example, setting the scope to `application` causes the same instance of the implementation to be used for each request. Setting the scope to `session` causes the same instance of the implementation to be used for each requests of a session. Setting the scope to `request` causes a new instance to be created for each 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* > **Web modules** > *module_instance*>**Web services implementation scope**.

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`.

Web Services Invocation Framework (WSIF): Enabling Web services

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 includes an EJB provider for EJB invocation using Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP). However, for EJB(IIOP)-based Web service invocation you should instead invoke RMI-IIOP Web services using JAX-RPC.

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 (EJB)) 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.

If you want to know more about the issues that WSIF addresses, see Goals of WSIF.

If you want to know how WSIF addresses these issues, see An overview of WSIF.

To use WSIF, see the following topics:
- Using WSIF to invoke Web services.
- WSIF system management and administration.
- WSIF API.

For more information about working with WSIF, visit the Web sites listed in Web services: Resources for Learning.

Goals of WSIF

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: As is explained in Web services are not just SOAP services, 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: 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. 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.

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.

An overview of WSIF

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 the goals of WSIF.

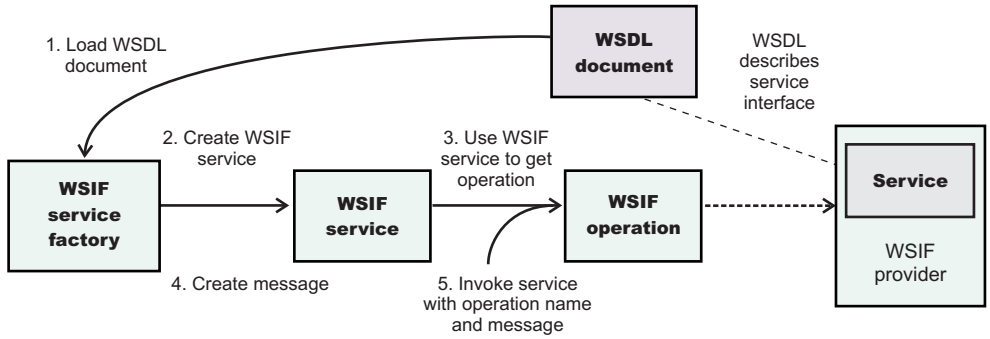
WSIF provides the following features:
- An API that provides binding-independent access to any Web service.
- A close relationship with WSDL, so it can invoke any service that you can describe in WSDL.
- A stubless and completely dynamic invocation of a Web service.
- The capability to plug a new or 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, see ″Running WSIF as a client″ and ″Interacting with the J2EE container in WebSphere Application Server″ in the information center. 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. See ″Using the Java Naming and Directory Interface (JNDI)″ in the information center.

For more conceptual information about WSIF and WSDL, see the following topics:
- WSIF and WSDL
- WSIF architecture
- Using WSIF with 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: 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 ″WSIF API reference: Finding a port factory or service″ in the information center.

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″ in the information center.

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 ″Using the WSIF providers″ in the information center.

Using WSIF with 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 on how to configure a client to dynamically select between multiple bindings, see ″Developing a WSIF service″ in the information center.

WSIF and WSDL: WSDL is the acronym for Web Services Description Language.

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 on 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: For the Web Services Invocation Framework (WSIF), dynamic invocation means providing the following levels of support when invoking Web services:

1. Support for WSDL extensions and bindings that were not known at build time.
2. Support for Web services that were not known at build time.

WSIF supports (1) through the use of providers. See ″Using the WSIF providers″ in the information center.

The providers support (2) by using the WSDL description to access the target service.

Getting started with 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.
- Managing the UDDI Registry explains how to use the UDDI pages in the WebSphere 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).

UDDI Registry troubleshooting might be useful if you encounter any problems or unexpected behaviour while using the UDDI Registry, and UDDI Registry messages explains any UDDI messages which 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 with to use a UDDI Registry.

UDDI Registry user interface tells you 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.

See ″Displaying the user interface″ for the URL for the UDDI User Interface.

″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 IBM 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 behaviour while using the UDDI Registry, and UDDI Registry messages explains any UDDI messages which you might see.

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.

Web services are Web applications that help you be more flexible in your business processes by integrating with applications that otherwise do not communicate.

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. To support Web services, extend WebSphere Application Server to support Web services standards. For interoperable Web services running on platforms supplied by multiple vendors, standards are essential.
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 Java bean or from Enterprise JavaBeans (EJB), 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 applications″ in the information center for information about developing Web services based on the Java language through the WebSphere Application Server. To read more about RAD see the information center for the product.
5. Install WebSphere Application Server. See ″Task overview: installing″ in the information center.
6. Review Web services Samples.

You have a design plan for implementing Web services applications into your business architecture.

Develop a Web service. See ″Developing Web services applications″ in the information center.

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 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

The Web services approach implements a service-oriented architecture (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**

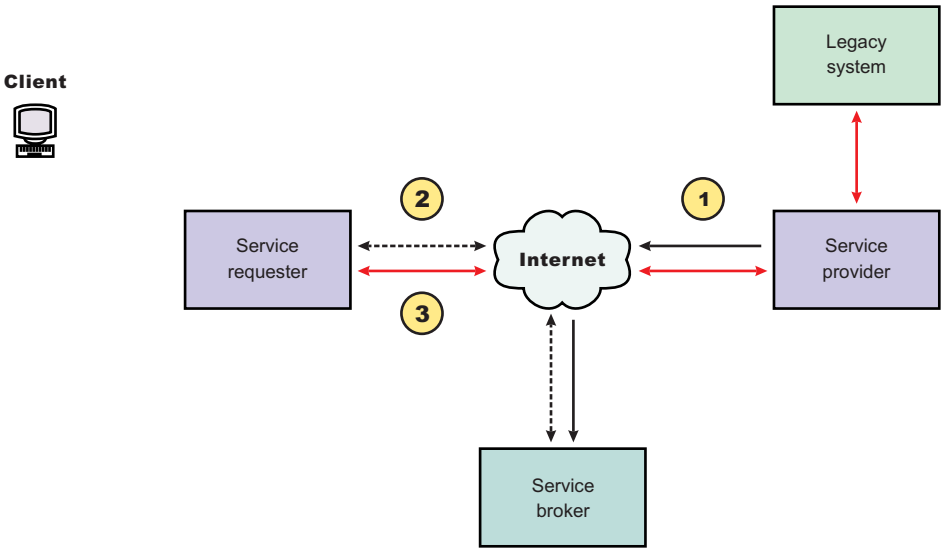
  The service provider creates a Web service and possibly publishes its interface and access information to the service registry. Each provider must decide which services to expose, how to make trade-offs between security and easy availability, how to price the services, or how to exploit free services 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.**

  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 service-oriented architecture 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.

Setting up and deploying a new UDDI Registry

Start the Installation of WebSphere Application Server and create the profile for your application server (for example, server1) to be used to host UDDI.

You have a choice of deploying either a default UDDI node, or a user customized UDDI node.

A default UDDI node would be a suitable option for initial evaluation of UDDI and for development and test purposes. 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 user customized UDDI node, you create the UDDI database and datasource to your own specifications, and then use the uddiDeploy.jacl script to deploy the UDDI application.

The main difference between *default* and *user customized*, in the context of these set up tasks, refers to a number of vital UDDI properties. For a default set up these vital properties are automatically set to default values and are not changeable by the user. For a user customizable set up, the user is given an opportunity to set these vital properties, but once set they cannot be changed for this configuration.

If you are setting up a UDDI node for production use, refer to Database considerations for production use of the UDDI Registry for information about choosing a database type.

The main guidance for deploying UDDI, elsewhere in this information center, is written for a single server configuration. This basic guidance applies to all configurations, however the following sections provide additional guidance for other variations such as:

- Network deployment
- Moving from a single server to a network deployment configuration
- Moving from a default to a customized node
- Moving between database types

If you are installing into a standalone application server you can proceed to either Setting up a default UDDI node, or Setting up a customized UDDI node.

Deploying into a Network Deployment cell (but not a cluster)

6.1+ The information in this section now applies to deploying into a cluster as well as a single server. For resources such as the JDBC provider and datasource, you can follow the guidelines for a non cluster configuration, however the resources may need updating on individual cluster members to correctly access the shared database for example.

It is important to note that the uddiDeploy.jacl script must be run with the Deployment Manager as the target.

If you are deploying into a network deployment cell you cannot create a default Cloudscape node using uddiDeploy.jacl. You may, however, manually create the default Cloudscape database (with a default profile) by following the instructions in Creating a Cloudscape database and adding the parameter 'DEFAULT' as the last argument.

You will not be able to use the default option on uddiRemove for a UDDI node that is deployed into an application server which is part of a Network Deployment cell. See Removing a UDDI Registry node for more information.

If you have a non default UDDI setup in a base application server, you can issue an *addNode -includeapps* command which will add the necessary definitions into the deployment manager.

Deploying into a cluster within a Network Deployment cell

6.1+ The scripts uddiDeploy.jacl and uddiRemove.jacl can now be used in a cluster environment, so the information in this section can be ignored.

It is important to note that uddiDeploy.jacl and uddiRemove.jacl cannot be used in a cluster environment.

It is assumed a single database will be used for all members of the cluster.

You can follow the same guidelines for a non cluster set up in general for the resources such as the JDBC provider and datasource as described in this Information Center, but they may need updating on individual cluster members to correctly access the shared database for example.

The options that are available are:

- Deploy UDDI into a standalone server, as described in this Information Center (using uddiDeploy), and then create a cluster using that server as a template for the other members.
- If using an cluster that already exists, you can use the admin console or wsadmin for example (and not uddiDeploy.jacl), to deploy the uddi.ear across the cluster members, but follow the additional advice in the main instructions.

Changing from a base application server to a Network Deployment cell.

It is possible to move from a base application server to a network deployment cell using the standard *addNode* command. During the move, any applications may be lost unless you use the *-includeapps* option. This applies to all applications and not just UDDI applications. See ″addNode command″ for details. This applies to a default or a customized UDDI node.

Moving from a default UDDI node to a user customized UDDI node.

After testing the UDDI deployment in a default UDDI node, you can move to a user customized node by recreating the database using the instructions in Setting up a customized UDDI node, but you must be aware that any data saved in the default node (policies, properties and user data) will be lost in the move.

Moving between Cloudscape and DB2 databases.

If you decide to move between one type of database to another, the datasource of the old database will still have a JNDI name of datasources/uddids. You must either rename this JNDI name to something different, or delete the datasource before you define the new database, create the new datasource and initialize the database.

You now have the choice of Setting up a default UDDI node orSetting up a customized UDDI node.

Database considerations for production use of the UDDI Registry

The IBM UDDI Registry fully supports a number of databases 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 or Oracle.

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 customized UDDI node

You can set up a customized UDDI node by completing the following steps:

1. Create a database schema to hold the UDDI registry by executing one of the following, ensuring that you do not perform the final step to set the default node indicator in the database:
   - Creating a DB2 database
   - Creating a Cloudscape database
2. Create a J2C Authentication Data Entry (not required for embedded Cloudscape, but required for network Cloudscape):
   a. Expand **Security**, **Global Security** and **JAAS Configuration** (on the right), then click **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), which is used to read and write to the UDDI registry database. For network Cloudscape the userid can be any value.

      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.
3. 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 datasource
Embedded Cloudscape	Cloudscape JDBC Driver	Connection Pool datasource
Network Cloudscape	Cloudscape Network Server Using Universal JDBC Driver	Connection Pool datasource

For details on how to create a JDBC provider, see either Using a DB2 Universal JDBC Driver Provider with WebSphere Application Server for z/OS, if you are using DB2 (do not follow the instructions for

creating a datasource; this is covered within the current task), orCreating and configuring a JDBC provider using the administrative console, for other database types.

4. Create a datasource for the UDDI Registry by following these steps:
   a. Expand **Resources** and **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 Version 4 option).
   e. Click **New** to create a new datasource.
   f. Fill in the details for the datasource as follows:

      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 datasources using this JNDI name. If you have another datasource 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 datasource and the UDDI application instance, before continuing.

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

      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.DB2UniversalDataStore Helper
Embedded Cloudscape	com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper
Network Cloudscape	com.ibm.websphere.rsadapter.CloudscapeNetworkServer DataStoreHelper

      Component-managed authentication alias
      - for **DB2** 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).

      Container-managed authentication alias
      set to (none)

      Mapping-configuration alias
      set to DefaultPrincipalMapping

      Relational Database Management System **data source properties**
      - for **DB2**:

Database name
this is the local LOCATION value. To find this value, enter the following operator command using the DB2 interactive panels:

```
-DIS DDF
```

Driver type
set to 2

- for **Cloudscape** (embedded or network) - **Database name** - for example:

```
install_root/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.

Click **Apply** and Save the changes to the master configuration.

5. Test the connection to your UDDI database by selecting the check box next to the datasource 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.
6. Deploy the UDDI Registry by running the wsadmin script uddiDeploy.jacl, as shown, from the *install_root*/bin directory.

   The syntax of the command is as follows:

```
wsadmin.sh [-conntype none] -f uddiDeploy.jacl
              node_name
              server_name
```

   6.1+

```
wsadmin.sh [-conntype none] -f uddiDeploy.jacl
              {node_name server_name | cluster_name}
```

   where

   - '-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 that the server name entered is case sensitive.
   - 6.1+ *cluster_name* is the name of the target cluster into which you wish to deploy the UDDI registry. Note that the cluster name entered is case sensitive.

   You are recommended to deploy the UDDI application using the uddiDeploy.jacl script, but note that you can also use the administrative console to deploy the application in the normal way. If you use the administrative console you must ensure that the Classloader Mode for the application is set to PARENT_LAST, and that the WAR class loader Policy is set to Application. The uddiDeploy.jacl script in a command prompt will do this for you.

   For example, to deploy UDDI on node 'MyNode' and server 'server1' (assuming that server1 is already started):

```
wsadmin.sh -f uddiDeploy.jacl MyNode server1
```

   6.1+ To deploy UDDI into cluster 'MyCluster' :

```
wsadmin.sh -f uddiDeploy.jacl MyCluster
```

7. Start the UDDI application, or start the application server if it is not already running. This will activate the UDDI node.

   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” for details).

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

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
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 Managing the UDDI Registry.
5. Once the properties have been set to appropriate values, click **Apply** to save your changes. It is important to save the changes before proceeding to the initialize step.
6. 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.

If the node has been migrated 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, you are now ready to use the UDDI Registry.

Setting up a default UDDI node

There are two ways to setup a default UDDI node:

- Either by executing the `uddiDeploy.jacl` script and specifying the **default** option. This creates a running default UDDI node within an embedded Cloudscape database, or
- Executing an additional step after you have created your database.

Run one of the following

1. **Optional: For a default Cloudscape UDDI node:**

   For a default Cloudscape network deployment configuration, read the section Installing into a Network Deployment Cell first.

   Deploy the UDDI Registry by running the wsadmin script uddiDeploy.jacl, as shown, from the *install_root*/bin directory.

   The syntax of the command is as follows:

```
wsadmin.sh [-conntype none] -wsadmin_classpath install_root/cloudscape/lib
               -f uddiDeploy.jacl
               node_name
               server_name
               default
```

   where

   - '-conntype none' is optional, and is only needed if the application server is not running.
   - *install_root* is the directory name of the WebSphere Application Server install 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 that 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, in a standalone application server. Note that this option cannot be used in a network deployment configuration.

For example, to create a UDDI node called 'MyNode' on server 'server1', you might enter the following (this assumes server1 is started):

```
wsadmin.sh -wsadmin_classpath /WebSphere/V6R0M0/AppServer/cloudscape/lib -f uddiDeploy.jacl MyNode server1 default
```

(Note that these should be entered as one command on a single line)

You should now start the UDDI application, or start the application server if it is not already running. This will activate the UDDI node.

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.

2. **Optional: For a default DB2 or default Cloudscape UDDI node:**

   DB2 or Cloudscape may be used for a single application server installation or a Network Deployment installation, including a cluster configuration. For the cluster configuration you must use network Cloudscape, as embedded Cloudscape is not supported for this scenario.

   a. Create a database schema to hold the UDDI registry by executing one of the following, ensuring that you perform the final step to set the default node indicator in the database:
      - Creating a DB2 database
      - Creating a Cloudscape database

   b. Create a J2C Authentication Data Entry (not required for embedded Cloudscape, but required for network Cloudscape):
      1) Expand **Security**, **Global Security** and **JAAS Configuration** (on the right), then click **J2C Authentication Data**.
      2) Click **New** to create a new J2C authentication data entry
      3) Fill in the details as follows:

         Alias a suitable (short) name, such as ″UDDIAlias″

         Userid
         the database userid (such as db2admin for DB2), which is used to read and write to the UDDI registry database. For network Cloudscape the userid can be any value.

         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.

   c. 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 datasource
Embedded Cloudscape	Cloudscape JDBC Driver	Connection Pool datasource
Network Cloudscape	Cloudscape Network Server Using Universal JDBC Driver	Connection Pool datasource

   For details on how to create a JDBC provider, see either Using a DB2 Universal JDBC Driver Provider with WebSphere Application Server for z/OS, if you are using DB2 (do not follow the

instructions for creating a datasource; this is covered within the current task), orCreating and configuring a JDBC provider using the administrative console, for other database types.

d. Create a datasource for the UDDI Registry by following these steps:
   1) Expand **Resources** and **JDBC Providers**.
   2) Select the desired 'scope' of the JDBC provider created earlier. For example, select:

      `Server: yourservername`

      to show the JDBC providers at the server level.
   3) Select the JDBC provider created earlier.
   4) Under **Additional Properties**, select **Data Sources** (*not* the Data Sources Version 4 option).
   5) Click **New** to create a new datasource.
   6) Fill in the details for the datasource as follows:

      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 datasources using this JNDI name. If you have another datasource 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 datasource and the UDDI application instance, before continuing.

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

      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.DB2UniversalDataStoreHelper
Embedded Cloudscape	com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper
Network Cloudscape	com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper

      Component-managed authentication alias
      - for **DB2** 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).

      Container-managed authentication alias
      Set to (none)

      Mapping-configuration alias
      Set to DefaultPrincipalMapping

      Relational Database Management System **data source properties**
      - for **DB2**:

Database name
this is the local LOCATION value. To find this value, enter the following operator command using the DB2 interactive panels:

```
-DIS DDF
```

Driver type
set to 2

- for **Cloudscape** (embedded or network) - **Database name** - for example:

  install_root/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.

Click **Apply** and Save the changes to the master configuration.

e. Test the connection to your UDDI database by selecting the check box next to the datasource 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.

f. Deploy the UDDI Registry by running the wsadmin script uddiDeploy.jacl, as shown, from the *install_root*/bin directory.

The syntax of the command is as follows:

```
wsadmin.sh [-conntype none] -f uddiDeploy.jacl
              node_name
              server_name
```

6.1+

```
wsadmin.sh [-conntype none] -f uddiDeploy.jacl
              {node_name server_name | cluster_name}
```

where

- '-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.
- 6.1+ *cluster_name* is the name of the target cluster into which you wish to deploy the UDDI registry. Note that the cluster name entered is case sensitive.

You are recommended to deploy the UDDI application using the uddiDeploy.jacl script, but note that you can also use the administrative console to deploy the application in the normal way. If you use the administrative console you must ensure that the Classloader Mode for the application is set to PARENT_LAST, and that the WAR class loader Policy is set to Application. The uddiDeploy.jacl script in a command prompt will do this for you.

For example, to deploy UDDI on node 'MyNode' and server 'server1' (assuming that server1 is already started):

```
wsadmin.sh -f uddiDeploy.jacl MyNode server1
```

6.1+ To deploy UDDI into cluster 'MyCluster' :

```
wsadmin.sh -f uddiDeploy.jacl MyCluster
```

g. Start the UDDI application, or start the application server if it is not already running. This will activate the UDDI node.

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 to use a default UDDI node, it will be initialized when the UDDI application is started for the first time.

Creating a DB2 database for the UDDI Registry

Perform this task if you want to use DB2 as the database store for your UDDI Registry data. You only need to do this once for each UDDI Registry, as part of Setting up and deploying a UDDI Registry.

1. Using the UNIX System Services (USS) command prompt, edit the createddl.sh script supplied in *install_root*/UDDIReg/rexx, 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.
2. Using the USS command prompt, run 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)
   ```

3. 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.

4. 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 for your UDDI Registry, and you do not want to use the default option on uddiDeploy.jacl (see 'Setting up a default UDDI node'). The most likely reasons for not using the default option on uddiDeploy.jacl are that you want to set up a customized UDDI node, or that you are deploying UDDI into a Network Deployment cell. You need only perform this task once for each UDDI Registry, as part of the Setting up and deploying a UDDI Registry.

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 install will be *install_root*/UDDIReg/databaseScripts

arg2 is the path to the location where you want to install the Cloudscape database, for example *install_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.

Run the following java -jar command from the *install_root*/lib directory, to create the UDDI Cloudscape database:

```
java -cp install_root/cloudscape/lib/db2j.jar -jar UDDICloudscapeCreate.jar arg1 arg2 arg3 arg4
```

Continue with setting up and deploying your UDDI Registry node.

Using a remote database for the UDDI Registry

It is possible for the UDDI Registry database to be hosted on a separate system (remote system) from the system on which the UDDI Registry application is deployed.

This is achieved using standard database capabilities of the database product used for the UDDI Registry database. You should refer to documentation for the database product if you are not familiar with these capabilities. Some considerations specific to each database product are:

Remote DB2

Create a DB2 UDDI database on the remote system, and use the DB2 Client to create an alias to reference it. Use the alias name as the Database name in the UDDI datasource.

Networked Cloudscape

Create a Cloudscape UDDI database on the remote system, and use the Cloudscape Network Server using Universal data source properties (Database name, Server name and Port number) of the UDDI datasource to reference the remote Cloudscape database.

For details of how to set up Cloudscape for multiple connections see Configuring Cloudscape Version 5.1.60x.

Note: Embedded Cloudscape is not supported for this configuration.

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 user interface″ in the information center).
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.

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 (EJBs) 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 - See ″Configuring endpoint URL information for HTTP bindings″ in the information center.
- Configure the URL endpoint information for JMS bindings.
- Configure the URL endpoint information to directly access enterprise beans. See ″Configuring endpoint URL information to directly access enterprise beans″ in the information center.

2. Externalize or publish the WSDL file out of the application. You can complete this task in one of three ways:
   - Publish a WSDL file with the administrative console. See ″Publishing WSDL files using the administrative console″ in the information center.
   - Publish a WSDL file through a URL. See ″Publishing WSDL files using a URL″ in the information center.
   - Publish a WSDL file with the **wsadmin** command tool. See ″Publishing WSDL files using the wsadmin tool″ in the information center.

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.

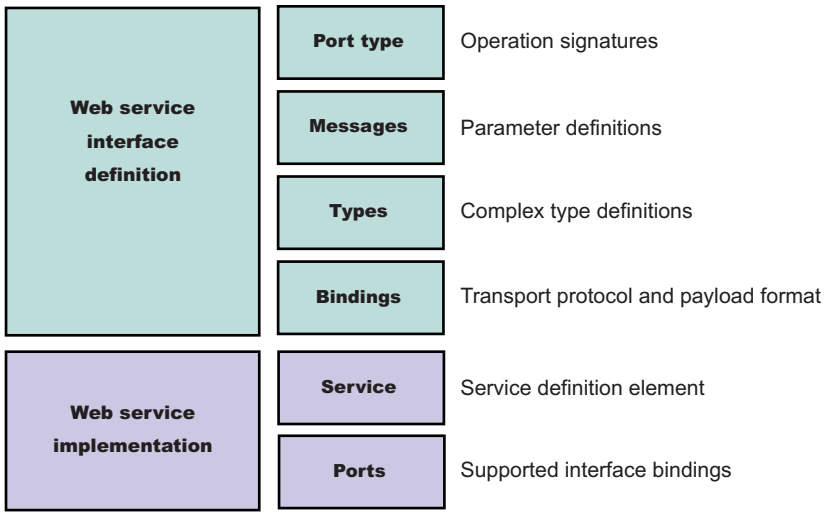
A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This separation supports the reuse of abstract definitions: messages, which are abstract 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 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″ in the information center.

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 JavaBean (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 API 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 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. See ″Starting the wsadmin scripting client″ in the information center.
2. Follow the steps in one of the following topics, depending on what task you want to complete:
   - Configure Web service client bindings. See ″Configuring a Web service client deployed WSDL file name with the wsadmin tool″ in the information center.
   - Configure Web service client preferred port mappings. See ″Configuring Web service client-preferred port mappings with the wsadmin tool″ in the information center.
   - Configure Web service client port information. See ″Configuring Web service client port information with the wsadmin tool″ in the information center.
   - Configure the scope of a Web service port. See ″Configuring the scope of a Web service port with the wsadmin tool″.

You have configured Web services applications with the wsadmin tool.

WSIF system management and administration

The Web Services Invocation Framework (WSIF) is provided as a stand-alone JAR file named `wsif.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 `wsif.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 on 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 (Web Services Invocation Framework) messages
- WSIF - Known restrictions.

Maintaining the WSIF properties file

The Web Services Invocation Framework (WSIF) properties are stored in the `wsif.jar` file, in a properties file named `wsif.properties`.

The wsif.jar file is located in the *install_root*/lib directory, where *install_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 wsif.jar file. Instead, you save the modified version in the *install_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. See ″Changing the default WSIF SOAP provider″ in the information center.

Enabling security for WSIF

The Web Services Invocation Framework (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, enable the following permission in the was.policy file: **FilePermission** to load the WSDL. This permission is required when a WSDL file is referred to using the file:// protocol.

Troubleshooting the Web Services Invocation Framework

For information on resolving WebSphere-level problems, see ″Diagnosing problems (using diagnosis tools)″ in the information center.

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 WSDL can cause numerous problems. To help ensure that your WSDL is valid, use a tool such as WebSphere Studio to create your service.

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 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, see the sample in ″Developing the WSIF client - the Address Book Sample″ in the information center.

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, see Enabling security for WSIF.

WSIF setup
Check that the `wsif.properties` file is correctly set up. For more information, see Maintaining the WSIF properties file.

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.

Here is a list of common errors, and information on their probable causes:

- **“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, see Enabling security for WSIF.

- **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 username 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.
```

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

To resolve the problem, you have three choices (in order of security, from least secure to most secure):

– Disable global security.

– For an equivalent level of security to the configuration on Version 5, modify the settings for the service integration bus that hosts the queue destination so that bus security is disabled and therefore the bus does not inherit security characteristics from the server or cell.

– For a greater level of security than the configuration on Version 5, configure basic authentication on each client that uses the service.

To disable global security, refer to either of these topics:

– Global security settings.

– Enabling and disabling global security using scripting.

To disable bus security, use the administrative console to complete the following steps:

1. Navigate to **Service Integration** → **Buses [Content Pane]** → **your_bus**.
2. Clear the **Secure** check box.
3. Save your changes.

To configure basic authentication on each client, use either the administrative console or the `wsadmin` tool. To complete the task using the `wsadmin` tool, see ″Configuring Web service client port information with the wsadmin tool″ in the information center and use the `WebServicesClientBindPortInfo wsadmin` task option. To complete the task using the administrative console, complete the following steps:

1. Navigate to **Applications** → **Enterprise Applications** → **application_instance** → **Web Modules or EJB Modules** → **module_instance** → **Web services: Client security bindings**.
2. Click **HTTP basic authentication** to access the “Configuring HTTP basic authentication with the administrative console” panel.
3. Enter the values in the panel.
4. Save your changes.

- **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. To enable interoperation, modify either your Web service or the WSIF default SOAP provider as described in ″WSIF SOAP provider: working with legacy applications″ in the information center.

Trace and logging for WSIF:

If you want to enable trace for the Web Services Invocation Framework (WSIF) API within WebSphere Application Server, and have trace, stdout and stderr for the application server written to a well-known location, see and ″Setting up component trace (CTRACE)″.

WSIF offers trace points at the opening and closing of ports, the invocation of services, and the responses from services.

To trace the WSIF API, you need to specify the following trace string:

```
wsif=all=enabled
```

WSIF also includes a `SimpleLog` utility through which you can run trace when using WSIF outside of WebSphere Application Server. To enable this utility, 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` uitility 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.

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

Welcome to the IBM WebSphere UDDI Registry.

Use the table of contents (on the left and below) to view the various topics for a specific product or technology. Select the topic you are interested in to either open documentation locally or find information about how to locate documentation.
- An Overview of the IBM Version 3 UDDI Registry
- UDDI Registry terminology
- Getting started with UDDI Registry
- Migrating to Version 3 of 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
- IBM JAXR Provider for the UDDI Registry
- UDDI Registry troubleshooting
- UDDI Registry Messages
- UDDI Registry Samples

An overview of the IBM 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.

IBM WebSphere UDDI Registry

The IBM WebSphere UDDI Registry is a directory for Web services that is implemented using the UDDI specification. It is a component of WebSphere Application Server Version 6.

A critical component of IBM's on-demand Service Oriented Architecture, the IBM WebSphere 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 IBM WebSphere 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 IBM WebSphere UDDI Registry was the first version 2 standard-compliant UDDI registry for private enterprise work. The IBM WebSphere UDDI Registry in WebSphere Application Server Version 6.0 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 WebSphere Application Server Version 6 are as follows (there are also some additional capabilities provided by the IBM WebSphere UDDI Registry in WebSphere Application Server Version 6.0 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 WebSphere Application Server Version 6.0 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 Admin Console extension

The WebSphere Application Server Version 6 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. In WebSphere Application Server Version 6 the databases that the UDDI Registry supports are DB2 and Cloudscape. Support is provided for remote access to these databases.

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 IBM 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 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 6.0 provides a JAXR Provider for accessing the IBM WebSphere 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 *install_root*. The default location is /WebSphere/V6R0M0/AppServer/.

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 specify the 'default' option when you run the uddiDeploy.jacl script, in which case the UDDI database will be a Cloudscape database. The second is to make sure the PDS member INSERT is included in the JCL used to create the database, in which case the UDDI database can be Cloudscape or DB2.

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 IBM WebSphere 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, 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 IBM WebSphere UDDI Registry in WebSphere Application Server Version 6.0 only supports single-node UDDI registries.

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 IBM WebSphere 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 IBM WebSphere 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 Version 6.0. 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 IBM UDDI Registry relate to the relevant OASIS specification and WebSphere Application Server level:

IBM 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 and later
2.1.x	• UDDI Version 1 • UDDI Version 2	5.1.x and later
3.0.2	• UDDI Version 1 • UDDI Version 2.0.4 (APIs), Version 2.0.3 (data structures) • UDDI Version 3.0.2	6.0.1

UDDI Registry user interface

This topic describes the UDDI user interface (also referred to as the UDDI User Console), which you can use to explore the IBM WebSphere UDDI Registry.

For information about how to display the UDDI user console, see Displaying the user interface.

If you will be using the UDDI console, you must configure the application server into which you have installed the UDDI Registry for UTF-8 encoding support: To do this, refer to ″Configuring application servers for UTF-8 encoding″ elsewhere in this Information Center.

- 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 users to issue inquiry requests and to familiarize themselves with general UDDI concepts. This section documents those areas for which support through the user console is not provided, together with 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 WebSphere 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
- **Note:** The UDDI Version 3 specification states that when a tModel is deleted, it should not be physically deleted. This allows the tModel to be reinstated. One effect of this is that, if you delete a tModel using the UDDI user console, the tModel is still visible through the Show Owned Entities display. Deleted tModels are displayed with the word 'deleted' against their name.

The UDDI user console is split into three areas. At the top of the screen are three links: **Home**, **Find** and **Publish**. The relevant panels appear below this bar when the links are clicked.

Home Returns you to the IBM WebSphere UDDI Registry welcome page

Find Activates the Find tab on the frame below to the left

Publish
Similarly activates the Publish tab on the frame below to the left

Below the WebSphere UDDI Registry banner the screen is split into two parts. On the left are the Find and Publish tabs:

Find tab

The Find pane is in two parts. At the top, a **Quick Find** service is provided. There are three radio buttons to enable a choice of 'service', 'business' and 'technical model' finds. Below these radio buttons is a text entry field for entering the name to search for and, beneath this, a **Find** button to start the search. Comments are provided to show you the wildcard character. The results of clicking **Find** are shown in the detail frame to the right.

Beneath the Quick Find is a section for **Advanced Find** functions which enables you to choose which entity they want to perform an advanced search on. There are three links: **Find services**, **Find businesses** and **Find technical models**. Clicking one of these links displays the corresponding advanced search form in the frame to the right, where you can specify search criteria. To perform a Find, first enter search criteria and select the required Find Qualifiers. Then click **Find Services** (or **Find Businesses** or **Find Technical Models**) to initiate the Find operation. The **Categorizations** section contains a link (**Show category tree**) which displays the tree from which you can select categories (or taxonomies) defining the types of item to find according to various classification systems. This is shown in the left-hand frame. In the advanced search form there are two buttons to start the search (mid-way down and at the bottom).

The results of the search are displayed in the same detail frame.

Publish tab

The *Publish* link on the top banner activates the Publish tab in the navigation frame to the left. The Publish pane is split into three sections.

1. **Quick Publish Function**

   The top part is a **Quick Publish** section to allow you to publish a business or technical model by name only. There are two radio buttons to enable a choice of 'business' or 'technical model'. Below these radio buttons is a text entry field for entering the name to assign to the selected entity and, beneath this, a **Publish** button to publish the entity. The results of clicking **Publish** are shown in the detail frame to the right.
2. **Advanced Publish Functions**

   To publish an entity with more detail, such as with multiple names, descriptions and categories, use the **Advanced Publish** section below this. The comments below each link (**Add a business** and **Add a technical model**) describe individual functions. Clicking one of these links displays the corresponding advanced publish form in the detail frame where you may enter details about the entity they want to publish. Similarly the **Categorizations** section allows taxonomies to be shown in the left frame from which you can select categories.

   Following entry of the relevant details in the **Advanced Publish** section, click **Publish Business** to publish the business to the UDDI Registry.
3. **Registered Information**

   Below the Advanced Publish section is a **Registered Information** section which has a link to **Show Owned Entities** in order to show the businesses, services and technical models registered to the individual user. Clicking the **Show Owned Entities** link displays the **Show Owned Entities** page in the detail frame at the right. The **Show Owned Entities** page is organized in two sections: **Registered Businesses** and **Registered Technical Models**. Each section shows the number of registered items.

   Edit and Delete Businesses

   You can **Edit** or **Delete** businesses owned by you, by clicking the appropriate links in the **Actions** column.

   After editing a business you *must* click **Update Business** to save the changes in the UDDI Registry.

   To delete a Business select the **Show Owner Entities** link and click the **Delete** link shown next to the business.

   Adding a Service to a Business

Services are added to a business by clicking the **Add Service** link in the **Actions** column of the **Registered Businesses** section.

Enter the details of the new service and click **Add Service** to publish the service to the UDDI Registry.

Technical Models

Technical Models owned by you are shown in the bottom **Registered Technical Models** section. As for businesses, users can Edit or Delete technical models owned by them by clicking the appropriate links in the **Actions** column.

Note: Users should take note that deletion of Technical Models (tModels) does **not** cause them to be physically deleted, but hidden. This is in accordance with the UDDI Registry Version 3.0 specifications. After deletion Technical Models are shown under the ″**Shown Owned Entities**″ link on the publish page but not via the Find links on the Find page. ALL other entities are deleted from the UDDI Registry in the normal way.

Example of publishing a Business, Service and tModel with the User Console

For the example, here, we will assume a business called Modern Cars that sells used cars

1. **Add the Business**

   Click the Publish tab in the left hand navigation frame. Then click 'Add a business' in the Advanced Publish in the left pane. This takes you to a 'Publish Business' pane on the right. Start by adding your Business Name in the text field labelled (Modern Cars in this example) and select a language and then click on the blue Add name to the right. This adds the business name. Below the Business Name is the descriptions field - free text can be added to describe the business. Enter a description and click the blue Add description link to add the description. You can add multiple descriptions in a variety of languages as required.

   Categorizations can be used to describe the business according to which categories it falls into. This example uses a Used car dealership. As an example, view the NAICS taxonomy by clicking 'Show category tree' and then expanding NAICS 2002 in the left hand panel. Expand the Retail Trade [44] entry by clicking the (+) plus sign next to it. You may need to drag the division between the left and right panes to be able to see all the category names. Similarly expand Motor Vehicle and Parts Dealers [441] then automobile Dealers [4411] and finally Used Car Dealers [44112]. Select 'Used Car Dealers [441120] and the 'Type', 'Key Name', 'Key Value' fields under categorizations will be filled in with the relevant values. Click the blue 'Add categorization' link to add the categorization details.

   Once all the fields are filled in, click **Publish Business** at the bottom of the form. A page is displayed showing the business details and the business is published to the UDDI Registry.

2. **Add a Service**

   Click the **Show owned entities** link to show the businesses in the UDDI Registry that are owned by you. In our Modern Cars example, to add the description of a service provided by the business click the **Add service** link and a Publish Service form is shown. At the top of the form in the Service Name field you can add a name, then select it's language and click the **Add name** link. You can also add a description (free text), one or more bindings (to add link points to the Service), and Categorizations (to add references to taxonomies to the service. After completing the required areas, clicking the 'Publish Service' button will Publish the service to the UDDI Registry with the current form contents.

3. **Adding a new technical model**

   Clicking the **Add a technical model** link in the left frame opens up the Publish Technical Model form on the right. Here you can enter the tModel name. Beneath this are the descriptions (a free text area to describe the technical model), overview documents (which gives a URL pointing to an overview document) and Categorizations (taxonomies describing the technical model). For each of these fields there is a blue Add link which must be clicked to add the relevant data. At the bottom of the form is a **Publish Technical Model** link which creates the technical model in the UDDI Registry.

UDDI Registry Management Interfaces

This topic explains interfaces and tools that you can use to manage UDDI nodes programmatically.

UDDI Registry Administrative (JMX) Interface

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.

UddiNodeProxy Usage

The following jars required for compilation can be found in *install_root*/lib/:

- admin.jar
- management.jar
- uddiadmin.jar
- wsexception.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.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
```

The nameKey and descriptionKey values can be used to look up the translated name and description for a given locale, using the messages.properties resource in uddiadmin.jar.

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 can be looked up in the messages.properties resource in uddiadmin.jar). While the PolicyGroup class does have a getPolicies

method it is important to note that PolicyGroup objects returned by the getPolicyGroups operation do not contain any Policy objects. This is so 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, you would 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 name (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 WebSphere 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(
                      "/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 IBM WebSphere UDDI Registry requires you to perform three tasks:

- publish a categorization tModel
- load the user defined value set data
- set the value set to **supported** status using the Administration console.

Only when all are complete will the checked value set be referenced. 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 Administration 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 Administration 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

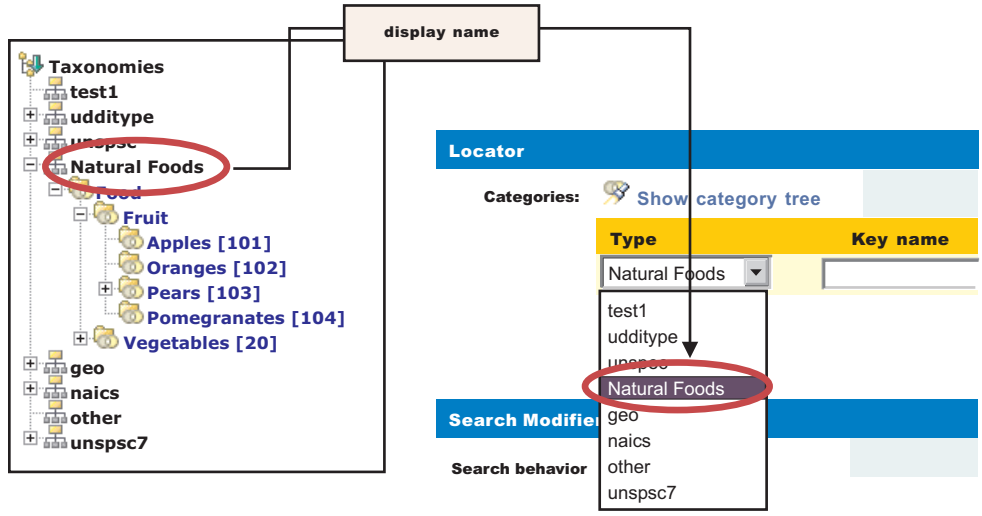
This section describes how to publish a checked categorization tModel with the **'Checked value set'** Key Name for use by a user defined value set.

Publish a tModel to the IBM WebSphere 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 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 IBM WebSphere 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.sh script is located in the *install_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 '%' sign as a column marker in the valuesetdata.txt file:

```
UDDIUserDefinedValueSet.sh -load valuesetdata.txt uddi:a708b8a7-35b5-451c-aafc-718ae071fcfe
-columnDelimiter %
```

Move value set data from one checked tModel to another on a UDDI Registry in a Network Deployment:

```
UDDIUserDefinedValueSet.sh –newKey uddi:a708b8a7-35b5-451c-aafc-718ae071fcfe
uddi:b819c9b8-46c6-562d-bb0d-829bf1820d0f –host depmanagerhost.ibm.com
–port 8879 –node uddinode –server uddiserver -override
```

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.sh –unload uddi:b819c9b8-46c6-562d-bb0d-829bf1820d0f
–server uddiserver –properties myproperties.properties
–password myrealpassword
```

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 Administration 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 'customTaxonomy: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

The UDDI Utility Tools are not supported on the z/OS platform; to use them you must install the Client Developer Kit for z/OS, on a non z/OS system, as described in Installing the Client Development Kit for z/OS. Alternatively, if you have another WebSphere Application Server installation on a non z/OS system, you can use the UDDI Utility Tools which are provided as part of that installation.

The UDDI Utility Tools can be used by running the UDDIUtilityTools.jar file. This file is located in the *install_root*/UDDIReg/scripts directory. Alternatively, all of the functions of UDDI Utility Tools can be invoked 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.

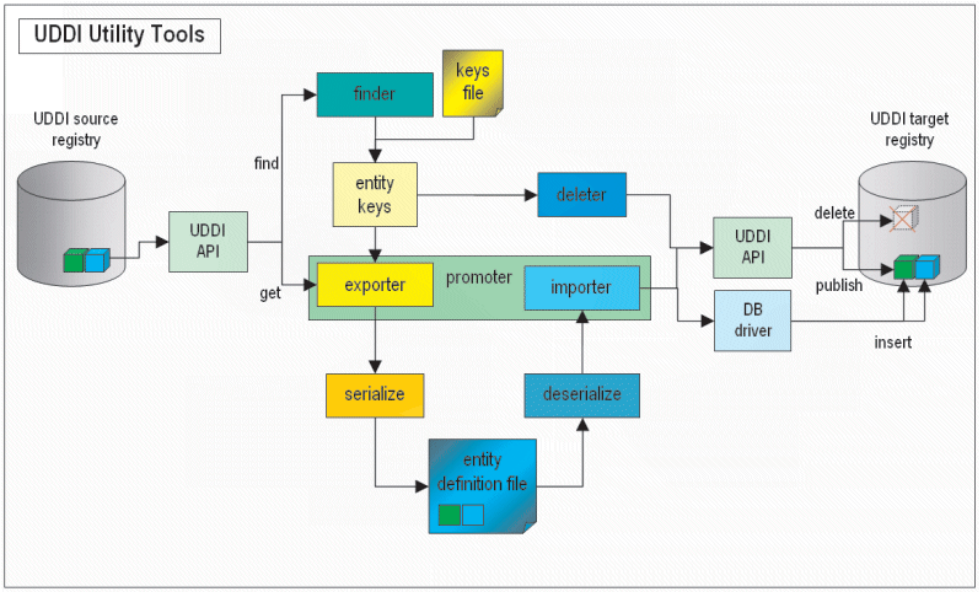
Delete Deletes the specified entities from the target UDDI Registry. The entities to delete are specified as an entity type, or a list of entity types, and keys, in the same way as for the export function.

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 *install_root*/UDDIReg/scripts directory. Copy this sample file to a user writable location, modify the file and specify this modified file when running the utility tools.

The most important property to set is the classpath, and this should include the current directory (.) and the UDDIUtilityTools.jar itself, plus all the dependent jars, most of which are located in the WebSphere AppServer lib directory. The classpath must include the database driver jar (for example db2java.zip). The other properties are well commented in the example properties file.

Below is an example properties file as distributed:

```
##########################################
# Runtime environment                    #
# (if invoking via java -jar...)         #
# "X Y" required around paths with spaces    #
# Replace WAS_HOME with your WAS home path.  #
# db2java.zip is for DB2 - replace this with #
# appropriate database driver file.          #
##########################################
classpath=.;WAS_HOME/UDDIReg/scripts/UDDIUtilityTools.jar;WAS_HOME/lib/soap.jar;
WAS_HOME/lib/uddi4jv2.jar;WAS_HOME/lib/j2ee.jar;
```

```
"C:/Program Files/IBM/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:uddi20
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=c:/websphere/appserver/etc/DummyClientTrustFile.jks

# The password for the trust store
trustStore.password=WebAS

############################################
# 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=C:/definitions/entities01.xml

# namespace prefix to use in definition file (export)
namespacePrefix=promote
```

Prerequisites

To run the UDDI Utility Tools you must use the IBM Development Kit for Java that is supplied with WebSphere Application Server. This Development Kit is located in *install_root*/java/bin. You must also 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 *install_root*/UDDIReg/scripts.

uddi4jv2.jar
This file contains the UDDI4J classes and is located in *install_root*/lib.

j2ee.jar
This file contains some required J2EE classes and is located in *install_root*/lib.

soap.jar
This is the Apache SOAP implementation and is located in *install_root*/lib.

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**
DBDriverLocation for classpath	*DB2_HOME*/jcc/classes/db2jcc.jar	*install_root*/cloudscape/ lib/otherJars/db2jcc.jar,*install_root* /cloudscape/lib/ db2jcc_license_c.jar
Driver	com.ibm.db2.jcc.DB2Driver	com.ibm.db2.jcc.DB2Driver
URL	jdbc:db2:*database_name*	jdbc:db2j:net://host: 1527/*database_name* (see note below)

where

- *install_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 your database. For Cloudscape, make sure that *database_name* includes the path to the database, for example *install_root*/profiles/AppSrv01/databases/com.ibm.uddi/UDDI30

If you are using Cloudscape, make the database network enabled so that it can handle multiple connections. See Configuring Cloudscape Version 5.1.60x for details on how to do this.

If you are using DB2, add DB2_HOME/jcc/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 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 by setting the PATH statement to the level of java supplied with WebSphere. For example, from the command line, type:

```
export PATH=install_root/java/bin:$PATH
```

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 entites 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 /uddikeys.txt
```

Below are a set of UDDI Utility Tools command line examples:

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 /myKeyFiles/keyFile01.txt
```

As above but also specifying verbose output to appear on the command line.

```
java -jar UDDIUtilityTools.jar -export -keysFile /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.

```
java -jar UDDIUtilityTools.jar -import -definitionFile /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 /UUT/myUUTProps.properties
```

To promote a set of entities specified in a keys file.

```
java -jar UDDIUtilityTools.jar -promote -keysFile /myKeyFiles/keyFile03.txt
```

As above but specifying that existing entities in the target registry get overwritten.

```
java -jar UDDIUtilityTools.jar -promote -keysFile /myKeyFiles/keyFile04.txt -overwrite
```

To promote a set of entities specified in a keys file including referenced tModels.

```
java -jar UDDIUtilityTools.jar -promote -keysFile /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 /myKeyFiles/keyFile06.txt
-definitionFile /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 /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 [263f5526-8708-4834-9f5d-8f8c878f5d6e].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with
 serviceKey [0af2a30a-be70-401f-a027-331a6c332712].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with
 serviceKey [61012761-d02c-4c70-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-4e81-88a7-e26bfbe2a7f1].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with
 serviceKey [003d2b00-f6c0-4071-8b84-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 [84d8e584-2510-4099-9b2a-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=/temp/MigToolFiles/Results/Promote_api_EDF_1.xml

ExportConfiguration:
  fromGetURL=http://yottskry:9080/uddisoap/
  fromInquiryURL=http://yottskry:9080/uddisoap/inquiryAPI

ImportConfiguration:
  overwrite=true
  uddiEntityDefinitionFile=/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=/WebSphere600/AppServer/etc/DummyClientTrustFile.jks
  secureConnection=false

DatabaseConfiguration:
  dbDriver=COM.ibm.db2.jcc.DB2Driver
  dbURL=jdbc:db2:LOC1
  dbUser=db2admin

LoggerConfiguration:
  messageStream=null
  messageLogFileName=/temp/MigToolFiles/logs/message.log
  traceLogFileName=/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 Javadoc welcome page in the information center topic, ″Reference: Generated 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, see Configuring Cloudscape Version 5.1.60x.

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.
- The JSSE provider class is not configurable; it must be com.ibm.jsse.IBMJSSEProvider.
- 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.

IBM 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 IBM UDDI 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 IBM 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 IBM WebSphere UDDI Registry for UDDI Version 3 in WebSphere Application Server Version 6.0 supports the UDDI Version 1, 2 and 3 SOAP APIs, and so the IBM JAXR Provider for UDDI can be used to access this

registry. The IBM JAXR Provider can also be used to access the IBM WebSphere 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 IBM 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 IBM 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 IBM 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 IBM JAXR Provider for UDDI are contained within the archive jaxruddi.jar, located in the *install_root*/lib directory. When using the JAXR API from within a J2EE application running under WebSphere Application Server Version 6.0, 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: bootstrap.jar, jaxruddi.jar, j2ee.jar, soap.jar and uddi4jv2.jar, which are all located in the *install_root*/lib directory.

javax.xml.registry.ConnectionFactory

To use the IBM 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 IBM 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 IBM JAXR Provider for UDDI to access the IBM WebSphere 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 IBM WebSphere UDDI Registry. The IBM JAXR Provider for UDDI does not define any additional provider-specific properties.

Property	Description

javax.xml.registry.queryManagerURL	The URL of the IBM WebSphere 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 IBM WebSphere 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 IBM 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 IBM 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 when WebSphere Global 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.

USING SSL (Secure Sockets Layer)

SSL can be used to encrypt HTTP traffic between the IBM JAXR Provider for UDDI and the IBM WebSphere 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 IBM WebSphere 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 IBM JVM in WAS 6.0, the IBM JSSE 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 WebSphere‘s 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 JVM prior to that in WAS 6.0, 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 in ″Introduction: Security″ within this Information Center.

Internal taxonomies

The IBM 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 IBM UDDI Version 3 Registry. If using the IBM JAXR Provider to access an IBM 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 geo-data.txt on the root of jaxruddi.jar. This is the taxonomy data file for the supplied ISO 3166 taxonomy. The first few lines are:

```
geo#--#World#--
geo#AE#United Arab Emirates#--
geo#AF#Afghanistan#--
geo#AG#Antigua And Barbuda#--
geo#AI#Anguilla#--
geo#AL#Albania#--
geo#AM#Armenia#--
geo#AN#Netherlands Antilles#--
geo#AO#Angola#--
geo#AQ#Antarctica#--
geo#AR#Argentina#--
geo#AR-A#Salta#AR
geo#AR-B#Buenos Aires#AR
```

   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 jaxruddi.jar.

      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 IBM 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.

UDDI Registry Messages

The UDDI Registry issues messages to report events or errors. The messages are in the form `CWUDxnnnns` where:

x is a character descriptor identifying which UDDI component is involved
nnnn is the error code
s is one of I (Information), E (Error) or W (Warning)

The prefix *CWUDxnnnns:* is followed by text that describes the event or error. For some messages, the first word of the text is one of the form (MSN=SSSS). The SSSS provides a message sequence number (or MSN), which identifies the unique circumstance in which the message was issued, and is of use where the same message can be issued in more than one circumstance.

To help you diagnose problems and minimize the need to enable trace in any of the above components, view the messages table. You can view the messages by prefix or component, whichever is easiest for you to find in the table. All messages are documented with user/system action and explanation.

The text for the UDDI messages is contained in the following files:

- *setupuddimessages.jar* and *UDDICloudscapeCreate.jar* for the CWUDD messages

- *jaxruddi.jar* for CWUDX messages
- *UDDIValueSetTools.jar* for CWUDV messages
- *UDDIUtilityTools.jar* for CWUDU messages
- *uddiresourcebundles.jar* for the remaining prefixes

which are placed, by the installation process, into the lib subdirectory of the WebSphere application server into which the UDDI Registry was installed (with the exception of UDDIUtilityTools.jar, which is placed into UDDIReg/scripts). If you will be running a console or log analyzer from another process; for example, to analyze the activity log, you must place a copy of the above jar files into a directory that is within the classpath of that process. Otherwise, the message lookup for the UDDI messages will fail.

UDDI Components Message Prefix Table **Click on individual links for message documentation for the component**	
CWUDDnnnns	UDDI Deploy and Removal
CWUDGnnnns	UDDI User Console
CWUDMnnnns	UDDI Management Interface
CWUDNnnnns	UDDI Node Management
CWUDQnnnns	UDDI Migration
CWUDRnnnns	UDDI Logging and Tracing
CWUDSnnnns	UDDI SOAP Interface
CWUDTnnnns	UDDI Transaction Manager
CWUDUnnnns	UDDI Utility Tools
CWUDVnnnns	UDDI Value Set Tools
CWUDXnnnns	JAXR

CWUDDnnnns (Web Services UDDI Deployment and Removal) messages:

CWUDD0001I: Attempting to deploy UDDI Registry application.

Explanation: The UDDI deployment process is starting.

User Response: None..

CWUDD0002W: Failed to discard unsaved changes caught exception Exc. Value is: <Exception Message>

Explanation: This warning message indicates that changes in the wsadmin session previous to running the uddiDeploy.jacl script cannot be discarded. The <Exception Message> describes the error that occurred.

User Response: None.

CWUDD0003I: Application Manager found.

Explanation: An active Application Manager can be used to stop and start UDDI on the deployment server.

User Response: None.

CWUDD0004I: Application Manager unavailable, application will not be started/stopped.

Explanation: No active Application Manager can be found so there is no need to stop and start UDDI on the deployment server.

User Response: None.

CWUDD0005I: Application Manager not running, application will not be started/stopped.

Explanation: Application Manager has been found, but it is not running so there is no need to stop and start UDDI on the deployment server.

User Response: None.

CWUDD0006I: Checking for installed UDDI Registry application of name appname. Value is: <UDDI Application Name>

Explanation: The deployment script is checking for deployed UDDI applications.

User Response: None.

CWUDD0007I: Stopping application of name appname. Value is: <Application Name>

Explanation: Attempting to stop a deployed UDDI application.

User Response: None.

CWUDD0008W: Stopping application appname caught exception Exc. Application might not have been running on this server. Values are: <Application Name> <Exception Message>

Explanation: The UDDI application stop request failed.

User Response: Attempt to stop the UDDI application via the Administration Console and rerun the uddiDeploy.jacl script.

CWUDD0009I: Application appname stopped successfully. Value is: <Application Name>

Explanation: The UDDI application has stopped successfully.

User Response: None.

CWUDD0010I: Removing application appname. Value is: <Application Name>

Explanation: The deployment script is attempting to remove an existing UDDI application.

User Response: None.

CWUDD0011I: Application appname removed successfully. Value is: <Application Name>

Explanation: The UDDI application has been removed from the configuration.

User Response: None.

CWUDD0012I: Attempting to create default UDDI datasource.

Explanation: A default UDDI deployment has been requested and therefore a Cloudscape datasource will be created.

User Response: None.

CWUDD0013I: Multiple Cloudscape JDBC Providers found. Using first in list.

Explanation: The deployment script has detected a number of suitable Cloudscape JDBC providers and will use the first.

User Response: None.

CWUDD0014I: UDDI Datasource name successfully created. Value is: <Datasource Name>

Explanation: A Cloudscape datasource has been created for the default UDDI deployment.

User Response: None.

CWUDD0015I: Setting application classloader mode.

Explanation: The UDDI application requires PARENT_LAST class loading.

User Response: None.

CWUDD0016I: Altered classloader mode of application appname from oldmode to newmode. Values are: <Application Name> <Old Classloader Mode> <New Classloader Mode>

Explanation: The UDDI application class loader has been successfully changed.

User Response: None.

CWUDD0017I: Setting classloader mode for application modules.

Explanation: The UDDI application web modules require PARENT_LAST class loading.

User Response: None.

CWUDD0018I: Altered classloader mode of module modname in application appname from oldmode to newmode. Values are: <Web Module Name> <Application Name> <Old Classloader Mode> <New Classloader Mode>

Explanation: The UDDI Web modules class loader mode has been successfully changed.

User Response: None.

CWUDD0019I: UDDI successfully deployed.

Explanation: UDDI has been successfully deployed.

User Response: None.

CWUDD0020I: Attempting to save new configuration.

Explanation: Attempting to save the configuration after removing any existing UDDI application.

User Response: None.

CWUDD0021I: Changes saved successfully.

Explanation: The configuration changes have been successfully saved.

User Response: None.

CWUDD0022I: Attempting to save new configuration.

Explanation: Attempting to save the configuration after installing the UDDI application ear.

User Response: None.

CWUDD0023W: Changes saved successfully.

Explanation: The configuration changes have been successfully saved.

User Response: None.

CWUDD0024I: Attempting to save new configuration.

Explanation: Attempting to save the configuration after changing the class loader modes.

User Response: None.

CWUDD0025I: Changes saved successfully.

Explanation: The configuration changes have been successfully saved.

User Response: None.

CWUDD1001W: Failed to discard unsaved changes caught exception Exc. Value is: <Exception Message>

Explanation: This warning message indicates that changes in the wsadmin session previous to running the uddiDeploy.jacl script cannot be discarded. The <Exception Message> describes the error that occurred.

User Response: None.

CWUDD1002I: Application Manager found.

Explanation: An active Application Manager can be used to stop and start UDDI on the deployment server.

User Response: None.

CWUDD1003I: Application Manager unavailable, application will not be started/stopped.

Explanation: No active Application Manager can be found so there is no need to stop and start UDDI on the deployment server.

User Response: None.

CWUDD1004I: Application Manager not running, application will not be started/stopped.

Explanation: Application Manager has been found, but is not running so there is no need to stop and start UDDI on the deployment server.

User Response: None.

CWUDD1005I: Stopping application of name appname. Value is: <Application Name>

Explanation: Attempting to stop a deployed UDDI application.

User Response: None.

CWUDD1006W: Stopping application appname caught exception Exc. Application might not have been running on this server. Values are: <Application Name> <Exception Message>

Explanation: The UDDI application stop request failed.

User Response: Attempt to stop the UDDI application via the Administration Console and rerun the uddiDeploy.jacl script.

CWUDD1007I: Application appname stopped successfully. Value is: <Application Name>

Explanation: The UDDI application has stopped executing.

User Response: None.

CWUDD1008I: Removing application appname. Value is: <Application Name>

Explanation: The removal script is attempting to remove the UDDI application.

User Response: None.

CWUDD1009I: Attempting to remove UDDI Registry application.

Explanation: The UDDI removal process is starting.

User Response: None.

CWUDD1010I: Application appname removed successfully. Value is: <Application Name>

Explanation: The UDDI application has been removed from the configuration.

User Response: None.

CWUDD1011I: Attempting to remove the default UDDI datasource.

Explanation: A default UDDI removal has been requested and therefore the Cloudscape datasource used for UDDI will be removed.

User Response: None.

CWUDD1012I: UDDI Datasource name successfully removed. Value is: <Datasource Name>

Explanation: The UDDI Cloudscape datasource has been removed from the configuration.

User Response: None.

CWUDD1013I: UDDI Datasource name does not exist. No action required. Value is: <Datasource Name>

Explanation: The default UDDI Cloudscape datasource does not exist in this configuration and therefore does not need to be removed.

User Response: None.

CWUDD1014I: UDDI Registry application and default UDDI datasource removed successfully.

Explanation: UDDI has been successfully removed from the configuration.

User Response: None.

CWUDD1015I: Attempting to save new configuration.

Explanation: Attempting to save the configuration after removing the UDDI application.

User Response: None.

CWUDD1016I: Changes saved successfully.

Explanation: The configuration changes have been successfully saved.

User Response: None.

CWUDD1017I: UDDI Registry application removed successfully.

Explanation: UDDI has been successfully removed from the configuration.

User Response: None.

CWUDD1018W: Application appname is not installed. Value is: <Application Name>

Explanation: The UDDI application has not been deployed and therefore the application cannot be removed.

User Response: None.

CWUDD3001I: Commencing UDDI Cloudscape database creation

Explanation: This is an informational message. It indicates that creation of the UDDI Cloudscape database is being started.

User Response: None

CWUDD3002I: Path to scripts='<DBscriptsLocation>'.

Explanation: This is an informational message. It indicates the location (path) of the scripts for creating the UDDI database, in the <DBscriptsLocation> insert.

User Response: None

CWUDD3003I: Path of database='<DBlocation>'.

Explanation: This is an informational message. It indicates the location (path) to be used for the UDDI Cloudscape database, in the <DBlocation> insert.

User Response: None

CWUDD3004I: Database name='<DBname>'.

Explanation: This is an informational message. It indicates the name to be used for the UDDI Cloudscape database, in the <DBname> insert.

User Response: None

CWUDD3005I: Default profile requested

Explanation: This is an informational message. It indicates that the UDDI Cloudscape database is to be created as a default UDDI database, using the default UDDI profile.

User Response: None

CWUDD3006I: Default profile not requested

Explanation: This is an informational message. It indicates that the UDDI Cloudscape database is to be created as a non-default UDDI database.

User Response: None

CWUDD3007I: Attempting to create or connect to UDDI Cloudscape database container

Explanation: This is an informational message. It indicates that an attempt is being made to create, or connect to, the database container for the UDDI Cloudscape database.

User Response: None

CWUDD3008I: UDDI Cloudscape database container successfully created or connected to

Explanation: This is an informational message. It indicates that the database container for the UDDI Cloudscape database has been connected to successfully.

User Response: None

CWUDD3009I: UDDI Cloudscape database creation completed normally

Explanation: This is an informational message. It indicates that the UDDI Cloudscape database has been successfully created.

User Response: None

CWUDD3010I: Attempting to open DDL File List file of name FileName. Value is: FileName='<DDLlistFilename>'

Explanation: This is an informational message. It indicates that the DDL File List file, which lists the DDL Files (database scripts) to be used to create the UDDI Cloudscape database, has been successfully opened. The name of the DDL File List file is given in the <DDLlistFilename> insert.

User Response: None

CWUDD3011I: Reading the contents of the DDL File List file and verifying

Explanation: This is an informational message. It indicates that the contents of the DDL File List file are being read and verified.

User Response: None

CWUDD3012I: Comment from file: <DDLfileComment>

Explanation: This is an informational message. It indicates that a comment line has been read from the DDL File List file, and shows the comment in the <DDLfileComment> insert.

User Response: None

CWUDD3013I: Line from file: <DDLfileLine>

Explanation: This is an informational message. It indicates that a non-comment line has been read from the DDL File List file, and shows the line in the <DDLfileLine> insert.

User Response: None

CWUDD3014I: Extraneous attributes will be ignored

Explanation: This is an informational message. It indicates that more attributes have been specified than are required, and that the extraneous attributes will be ignored.

User Response: None

CWUDD3015I: Skipping the Default Profile record because Default Profile was not requested

Explanation: This is an informational message. It indicates that the default UDDI profile was not requested, and that therefore the record in the DDL File List file for setting up the default profile will be skipped.

User Response: None

CWUDD3016I: Attempting to populate the database container with schema structures.

Explanation: This is an informational message. It indicates that an attempt is being made to add schemas to the UDDI Cloudscape database container.

User Response: None

CWUDD3017I: Attempting to load the Cloudscape JDBC driver.

Explanation: This is an informational message. It indicates that an attempt is being made to load the JDBC driver class for Cloudscape.

User Response: None

CWUDD3018I: Cloudscape JDBC driver successfully loaded

Explanation: This is an informational message. It indicates that the JDBC driver for Cloudscape has been successfully loaded.

User Response: None

CWUDD3019I: Processing DDL file DDLFile using Term as the terminator. Values are: DDLFile=<DDLfilename>, Term=<terminator>.

Explanation: This is an informational message. It indicates that the DDLFile whose name is in the <DDLfilename> insert is being processed, with the character in the <terminator> insert being used as terminator.

User Response: None

CWUDD3020I: DDL file successfully processed. N statements processed. Value is: N=<numStatements>.

Explanation: This is an informational message. It indicates that the current DDL file has been successfully processed, and that the number of statements indicated by the <numStatements> insert were processed.

User Response: None

CWUDD3021I: End of file reached

Explanation: This is an informational message. It indicates that the end of the current DDL file has been reached.

User Response: None

CWUDD3022I: Converting SQL string Str to Cloudscape syntax. Value is: Str=<SQLstring>.

Explanation: This is an informational message. It indicates that the SQL string given in the <SQLstring> insert is being converted into an SQL syntax that is recognized by Cloudscape.

User Response: None

CWUDD3030I: UDDI Cloudscape database successfully completed

Explanation: This is an informational message. It indicates that creation of the UDDI Cloudscape database has completed successfully.

User Response: None

CWUDD4001E: An Exception Exc occurred during creation of UDDI Cloudscape database. Value is:

Explanation: An exception occurred during the attempt to create the UDDI Cloudscape database.

User Response: The <exception> insert provides information that should help you diagnose the cause of the problem.

CWUDD4002E: Abnormal exit from creation of UDDI Cloudscape database.

Explanation: The attempt to create the UDDI Cloudscape database is exiting abnormally.

User Response: Examine the preceding messages to determine the reason for the abnormal exit.

CWUDD4003E: Insufficient arguments supplied

Explanation: Insufficient arguments have been supplied for creating the UDDI Cloudscape database.

User Response: Retry the request, supplying the correct number of arguments.

CWUDD4004E: Usage is: java -jar <thisjar> <arg1> <arg2> <arg3> <arg4>

where:

- **<thisjar> = name of jar file for creating UDDI Cloudscape Database**
- **<arg1> = path to DDL (SQL) files**
- **<arg2> = path to location for UDDI Cloudscape database**
- **<arg3> = name of UDDI Cloudscape Database**
- **<arg4> = (optional), if specified must be the string DEFAULT**

Explanation: This message gives the correct usage syntax for creating the UDDI Cloudscape Database. It is issued when the incorrect syntax has been used.

User Response: Correct the syntax to match the usage indicated by the message. You might also need to specify the Cloudscape class library on the classpath, by using the -cp parameter on the java command. Refer to the Information Center documentation on setting up and deploying UDDI for more details.

CWUDD4005E: Creation of UDDI Cloudscape database was unsuccessful

Explanation: The attempt to create the UDDI Cloudscape database has been unsuccessful.

User Response: Examine the preceding messages to determine the reason for this failure.

CWUDD4006E: SQL Exception Exc encountered during database container creation. Value is: Exc=<exception>.

Explanation: An SQL exception occurred when attempting to create the database container for the UDDI Cloudscape database.

User Response: The <exception> insert should contain information which will help you to diagnose the problem.

CWUDD4007E: DDL File List file not found

Explanation: The DDL File List file, used to specify the DDL files to be used to create the UDDI Cloudscape database, could not be found.

User Response: The DDL File List file should be included in the UDDICloudscapeCreate.jar used to create the UDDI Cloudscape database, so this error indicates a possible corruption of that jar file. Check that you have a valid version of UDDICloudscapeCreate.jar.

CWUDD4008E: Invalid attribute Attr found in DDL File List file. Value should be true or false. Value is: Attr=<attribute>.

Explanation: An invalid attribute was found in the DDL File List file. The expected attribute is one of 'true' or 'false'. The <attribute> insert indicates the value that was found.

User Response: The DDL File List file is included in the UDDICloudscapeCreate.jar used to create the UDDI Cloudscape database, so this error indicates a possible corruption of that jar file. Check that you have a valid version of UDDICloudscapeCreate.jar.

CWUDD4009E: Insufficient attributes found in DDL File List file.

Explanation: Insufficient attributes were found in the DDL File List file.

User Response: The DDL File List file should be included in the UDDICloudscapeCreate.jar used to create the UDDI Cloudscape database, so this error indicates a possible corruption of that jar file. Check that you have a valid version of UDDICloudscapeCreate.jar.

CWUDD4010E: SQL exception Exc encountered during database population. Value is: Exc=<exception>.

Explanation: An SQL exception has occurred while populating the UDDI Cloudscape database.

User Response: The <exception> insert contains the SQL exception which occurred. Use this to diagnose the problem.

CWUDD4011E: Delete existing UDDI Cloudscape database if it is to be overwritten with a new one.

Explanation: This message is issued when an exception has occurred while populating the UDDI Cloudscape database. A common cause of this problem is that a UDDI Cloudscape database already exists.

User Response: You can ignore this message if you want to keep the existing data in your existing UDDI Cloudscape Database. If you want to overwrite this with a new UDDI Cloudscape database, then you should delete the existing database, then and rerun the request. Previous messages will have shown the location and name of the UDDI Cloudscape database.

CWUDD4012E: Exception Exc occurred while trying to find the Cloudscape JDBC Provider. Value is: Exc=<exception>.

Explanation: An exception occurred while trying to find the Cloudscape JDBC driver class.

User Response: The <exception> insert contains the exception which occurred. Examine this to diagnose the problem.

CWUDD4013E: Ensure that the Cloudscape libraries are on the classpath

Explanation: This message is issued when an exception has occurred when trying to find the Cloudscape JDBC driver class. A common cause of this problem is that the classpath has not been set up to include the path to the Cloudscape class library.

User Response: Ensure that you have specified the Cloudscape class library on the request to create the Cloudscape UDDI database. Depending on how you issued the request, this might involve specifying the classpath on the wsadmin command used to invoke uddiDeploy.jacl, or on the java -jar command used to invoke the UDDICloudscapeCreate.jar file. Refer to the Information Center documentation on setting up and deploying UDDI for more details.

CWUDD4014E: Exception Exc occurred while processing SQL statement Str. Character positions within Str shown by StrPos.

Values are:

- **Exc=<exception>,**
- **<SQLstring>,**
- **StrPos=<charPositions>**

Explanation: An exception occurred while processing an SQL statement used to create the UDDI Cloudscape database. The message shows the SQL exception message in the <exception> insert, the SQL string which was being processed in the <SQLstring> insert, and a string of character positions in the <charPositions> insert.

User Response: Examine the exception message to determine the cause of the problem. If the exception message indicates the position at which the problem occurred, then use the <charPositions> string to identify that position in the SQLstring.

CWUDD4015E: There were no SQL statements in the DDL file.

Explanation: No SQL statements have been found in the DDL file that is currently being processed.

User Response: Previous messages will tell you the path to the database scripts (DDL files), and the DDL file which is currently being processed. Use this information to find the DDL file and check that it is valid.

CWUDD4016E: Exception Exc occurred while processing DDL file. Value is: Exc=<exception>.

Explanation: An exception occurred while processing the current DDL file.

User Response: Examine the exception message in the <exception> insert to diagnose the problem.

CWUDD4017E: Location of database scripts not specified.

Explanation: The request issued to create the Cloudscape database has not specified a location for the database scripts.

User Response: Retry the request, providing a location for the database scripts. If this message is issued when using the uddiDeploy.jacl script with the default option, then check that you are using a valid version of uddiDeploy.jacl.

CWUDD4018E: Location of database not specified.

Explanation: The request issued to create the Cloudscape database has not specified a location for the UDDI Cloudscape database.

User Response: Retry the request, providing a location for the UDDI Cloudscape database. If this message is issued when using the uddiDeploy.jacl script with the default option, then check that you are using a valid version of uddiDeploy.jacl.

CWUDD4019E: Name of database not specified.

Explanation: The request issued to create the Cloudscape database has not specified a name for the UDDI Cloudscape database.

User Response: Retry the request, providing a name for the UDDI Cloudscape database. If this message is issued when using the uddiDeploy.jacl script with the default option, then check that you are using a valid version of uddiDeploy.jacl.

CWUDD4020E: Invalid value specified for Default Profile parameter Parm, value should be GoodParm. Values are: Parm=<suppliedparm>, GoodParm=<expectedparm>.

Explanation: The request issued to create the Cloudscape database has specified an invalid value for the default profile parameter. The <suppliedparm> insert indicates the value that was supplied, and the <expectedparm> indicates the value that was expected.

User Response: Retry the request, specifying a valid value for the default profile parameter, or omit this parameter altogether if you do not want to create the UDDI Cloudscape database with a default profile. If this message is issued when using the uddiDeploy.jacl script with the default option, then check that you are using a valid version of uddiDeploy.jacl.

CWUDD4021E: An error occurred while loading the Cloudscape JDBC driver.

Explanation: An error occurred while attempting to load the Cloudscape JDBC driver class.

User Response: Examine the preceding messages for details of the error.

CWUDD6001E: Incorrect number of arguments passed to script.

Explanation: The wrong number of arguments were passed to the script. Arguments are Node Name, Server Name, and optionally the **default** keyword.

User Response: Retry with the correct arguments.

CWUDD6002E: Usage is: <Command format description>.

Explanation: The deployment script has not been called with the correct arguments.

User Response: Retry with the correct arguments.

CWUDD6003E: Failed to determine server ID caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to determine the Server ID.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6004E: Failed to determine server ID, possibly due to invalid nodename or servername (check case).

Explanation: The Server ID could not be located for the given node name and server name.

User Response: Check the node name and server name are correct and are in the correct case.

CWUDD6005E: Failed to determine the list of JDBC providers caught exception Exc. Value is: <Exception Message>.

Explanation: The deployment script was unable to determine the list of JDBC providers.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6006E: Failed to get JDBC provider name from id caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to determine the list of JDBC providers for the server.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6007E: Attempt to create a Cloudscape JDBC provider caught exception Exc. Value is: <Exception Message>.

Explanation: A Cloudscape JDBC provider is required for the default deployment of UDDI and an Exception occurred whilst trying to create this JDBC provider.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6008E: Failed to create datasource caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the UDDI Cloudscape datasource.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6009E: Failed to create resource property set caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to J2EEResourcePropertySet.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6010E: Failed to create 'databaseName' resource property caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the J2EE Resource Property 'databaseName'.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6011E: Failed to create 'shutdownDatabase' resource property caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the J2EE Resource Property 'shutdownDatabase'.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6012E: Failed to create 'datasourceName' resource property caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the J2EE Resource Property 'datasourceName'.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6013E: Failed to create 'description' resource property caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the J2EE Resource Property 'description'.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6014E: Failed to create 'connectionAttributes' resource property caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the J2EE Resource Property 'connectionAttributes'.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6015E: Failed to create 'createDatabase' resource property caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the J2EE Resource Property 'createDatabase'.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6016E: Attempt to locate the WebSphere Installation Directory failed.

Explanation: The installation directory of WebSphere Application Server could not be determined.

User Response: Verify that WebSphere Application Server has been correctly installed and that the **WAS_INSTALL_ROOT** WebSphere environment variable exists in the configuration.

CWUDD6017E: Uninstall of application appname caught exception Exc. Values are: <Application Name> <Exception Message>.

Explanation: An Exception occurred whilst trying to uninstall an existing UDDI application.

User Response: This message is self-explanatory.

CWUDD6018E: Install of UDDI application caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to install the UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6019E: Attempt to find application classloader failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to locate the classloader for the UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6020E: Attempt to read current classloader mode failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to determine the classloader mode of the UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6021E: Attempt to modify classloader mode to newmode failed with exception Exc. Values are: <Classloader Mode> <Exception Message>.

Explanation: An Exception occurred whilst trying to change the classloader mode of the UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6022E: Attempt to read new classloader mode failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to verify the new classloader mode of the UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6023E: Attempt to read module list from application failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to determine the list of modules held in the UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6024E: Attempt to locate URI attribute on module failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to determine the URI attribute of the application module.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6025E: Attempt to read current classloader mode from module failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to determine the classloader mode of a UDDI module.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6026E: Attempt to modify module classloader mode to newmode failed with exception Exc. Values are: <Classloader Mode> <Exception Message>.

Explanation: An Exception occurred whilst trying to change the classloader mode of a UDDI module.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6027E: Attempt to read new module classloader mode failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to verify the new classloader mode of a UDDI module.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6028E: Attempt to create default UDDI Registry Cloudscape database failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to create the default Cloudscape database.

User Response: Refer to messages displayed during creation to determine cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6029E: Attempt to locate the Nodes Variable Map failed with exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to locate the configurations Variable Map for the given node.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6030E: Error saving configuration, changes not saved due to exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to save the configuration after creating default datasource and removing an existing UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6031E: Error saving configuration, changes not saved due to exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to save the configuration after installing the UDDI application.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6032E: Error saving configuration, changes not saved due to exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to save the final configuration.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6033E: Incorrect argument passed to script.

Explanation: The wrong argument was passed to the script. Arguments are Node Name, Server Name, and optionally the **default** keyword.

User Response: Retry with the correct arguments.

CWUDD6034E: 'default' keyword is not allowed in a WebSphere Application Server Network Deployment configuration.

Explanation: A default Cloudscape UDDI Registry is not permitted in a WebSphere Application Server Network Deployment configuration. Please follow the UDDI Installation instructions on how to create a non default UDDI Registry database.

User Response: Retry with the correct arguments.

CWUDD6035E: Cluster names are only allowed in a WebSphere Application Server Network Deployment configuration.

Explanation: UDDI can only be deployed to a Cluster in a Network Deployment configuration.

User Response: Deploy UDDI using the arguments Node and Server.

CWUDD6036E: Failed to determine cluster ID caught exception Exc. Value is: <Exception Message>

Explanation: An Exception occurred whilst trying to determine the Cluster ID.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD6037E: Failed to determine cluster ID, possibly due to invalid clusterName (check case).

Explanation: The Cluster ID could not be located for the given cluster name.

User Response: Check the cluster name is correct and is in the correct case.

CWUDD7001E: Incorrect number of arguments passed to script.

Explanation: The wrong number of arguments were passed to the script. Arguments are Node Name, and optionally the **default** keyword.

User Response: Retry with the correct arguments.

CWUDD7002E: Usage is: <Command format description>.

Explanation: The removal script has not been called with the correct arguments.

User Response: Retry with the correct arguments.

CWUDD7003E: Failed to determine server ID caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to determine the Server ID.

User Response: Retry the removal of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD7004E: Failed to determine server ID, possibly due to invalid nodename or servername (check case).

Explanation: The Server ID could not be located for the given node name and server name.

User Response: Check the node name and server name are correct and are in the correct case.

CWUDD7005E: Uninstall of application appname caught exception Exc. Values are: <Application Name> <Exception Message>.

Explanation: An Exception occurred whilst trying to uninstall the UDDI application.

User Response: Retry the removal of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD7006E: Failed to remove default UDDI datasource caught exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to remove the UDDI Cloudscape datasource.

User Response: Retry the removal of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD7007E: Error saving configuration, changes not saved due to exception Exc. Value is: <Exception Message>.

Explanation: An Exception occurred whilst trying to save the final configuration.

User Response: Retry the removal of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD7008E: Incorrect argument passed to script.

Explanation: The wrong argument was passed to the script. Arguments are Node Name, Server Name, and optionally the **default** keyword.

User Response: Retry with the correct arguments.

CWUDD7009E: 'default' keyword is not allowed in a WebSphere Application Server Network Deployment configuration.

Explanation: A default Cloudscape UDDI Registry is not permitted in a WebSphere Application Server Network Deployment configuration. Therefore there is no default Cloudscape UDDI datasource to remove.

User Response: Retry with the correct arguments.

CWUDD7010E: Cluster names are only allowed in a WebSphere Application Server Network Deployment configuration.

Explanation: UDDI can only be removed from a Cluster in a Network Deployment configuration.

User Response: Remove UDDI using the arguments Node and Server.

CWUDD7011E: Failed to determine cluster ID caught exception Exc. Value is: <Exception Message>

Explanation: An Exception occurred whilst trying to determine the Cluster ID.

User Response: Retry the deployment of UDDI. If the error persists, examine the Exception information on its cause. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDD7012E: Failed to determine cluster ID, possibly due to invalid clusterName (check case).

Explanation: The Cluster ID could not be located for the given cluster name.

User Response: Check the cluster name is correct and is in the correct case.

CWUDGnnnns (Web Services UDDI User Console) messages:

CWUDG0001I: IBM WebSphere UDDI Registry user console starting initialization.

Explanation: The user console control servlet is starting.

User Response: None.

CWUDG0002I: IBM WebSphere UDDI Registry user console finished initialization.

Explanation: The user console control servlet has completed startup successfully.

User Response: None.

CWUDG0003I: Reading init parameters.

Explanation: The user console control servlet has started reading external parameters in its init method

User Response: None.

CWUDG0004I: Finished reading init parameters.

Explanation: The user console control servlet has finished reading external parameters in its init method. This message indicates the user console is ready to accept client requests.

User Response: None.

CWUDG0005E: A serious error has occurred. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: This error message indicates an unexpected error has occurred. The <Message> describes the error that has occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: A trace of the gui component is recommended. Contact IBM support with this information.

CWUDG0006E: A persistence error has occurred. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: An error occurred while performing a database operation. The <Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Check database connections and state. Provide IBM support with a trace, including the gui and persistence components.

CWUDG0007E: A User mismatch error has occurred. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: The user id provided does not match the user id required or expected whilst performing an operation that requires authentication. The <Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Check the user has authority for the operation being requested. If necessary, contact IBM support detailing the actions taken to recreate the problem.

CWUDG0008E: An invalid key was passed. Error message: <Message> error: <Throwable>. More information: <Additional information>.

Explanation: The requested operation is trying to retrieve information about an entity with a key that is invalid. This may occur if the entity has been deleted by another session. The <Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Ask the client to close existing sessions and attempt the operation in a new browser session. If the problem persists, contact IBM support with a trace of the gui and api components.

CWUDG0009E: An invalid value was supplied. Error message: <Message> error: <Throwable>. More information: <Additional information>

Explanation: An invalid value was passed to an API call. The <Message> describes the error that occurred and the <Throwable> is the type of error that was caught. <Additional information> may provide further information, if available.

User Response: Contact IBM support with a trace of the gui and api components.

CWUDG0010E: Failed to introspect ActionForm properties. Exception: <Exception>.

Explanation: String properties of a form object could not be introspected which means that the form contents cannot be checked for invalid characters.

User Response: Contact IBM support with details of the Exception and a trace of the gui component.

CWUDG0011E: Failed to invoke reflected methods in ActionForm. Exception: <Exception>.

Explanation: A form object's declared public method for setting or getting a String value could not be invoked. This method is required to check for invalid characters.

User Response: Contact IBM support with details of the Exception and a trace of the gui component.

CWUDG0012E: User console initialization failed to connect to UDDI database. Exception: <Exception>.

Explanation: During user console initialization, connection to the database failed, and threw the exception specified.

User Response: Check the connection to the UDDI database. The included exception message may yield some clues to help you resolve the problem. If unresolved, contact IBM support with a trace of the gui component during startup.

CWUDG0013E: User console initialization failed to initialize tModels. Exception: <Exception>.

Explanation: Indicates that an error has occurred during initialization of ActionServlet, specifically when reading tModels (invoking init method in class TModelNames).

User Response: Check the state of the UDDI database. Visually inspect the TMODEL table and confirm it is populated with valid data. The included exception message may yield some clues to help you resolve the problem. If unresolved, contact IBM support with a trace of the gui component during startup.

CWUDG0014E: User console initialization failed to initialize taxonomies. Exception: <Exception>.

Explanation: Indicates that an error has occurred during initialization of ActionServlet, specifically when reading taxonomy data (invoking init method of CategoryTaxonomyTree).

User Response: Check the state of the UDDI database. The included exception message may yield some clues to help you resolve the problem. If unresolved, contact IBM support with a trace of the gui component during startup.

CWUDMnnnns (Web Services UDDI Management Interface) messages:

CWUDM0001E: Unexpected error in MBean operation: <operation name>

Explanation: An internal error occurred processing the specified UddiNode MBean operation.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0002E: MBean transaction failed. Commit flag was: <truelfalse>

Explanation: Failed to commit or rollback the current transaction.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0003E: MBean transaction did not begin.

Explanation: Failed to invoke begin method on the UserTransaction.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0004E: MBean transaction connection failed to release.

Explanation: Failed to release connection after transaction committed or rolled back.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0005E: MBean could not establish control with persistence manager.

Explanation: Message is self-explanatory.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0006E: MBean could not acquire connection for UDDI datasource.

Explanation: Message is self-explanatory.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0007E: UddiNode MBean with ObjectName <ObjectName value> could not be recognized.

Explanation: The UddiNode MBean for the UDDI application could not be unregistered from the MBeanServer.

User Response: Restart the UDDI application or the application server. If the problem cannot be resolved, contact your IBM Customer Service Center, supplying the ObjectName value reported in the message (if present).

CWUDM0008W: MBean notification for event <notification identifier> failed.

Explanation: Message is self-explanatory.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0009W: A UddiNode MBean is already registered.

Explanation: There is a UddiNode MBean registered in the same cell, node and server.

User Response: Ensure there is only one UDDI application deployed in the server. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0020E: Unable to retrieve UDDI node ID.

Explanation: A database error occurred.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0021E: Unable to retrieve UDDI node state.

Explanation: A database error occurred.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0022E: Unable to retrieve UDDI node application name.

Explanation: A database error occurred.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0023E: Unable to retrieve UDDI node description.

Explanation: A database error occurred.

User Response: Check database connectivity and UDDI application installation configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0024E: Unable to activate UDDI node.

Explanation: An error occurred trying to activate the UDDI node.

User Response: Check the running state of the UDDI application. Restart the application if necessary. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0025I: Unable to activate a UDDI node that is not initialized.

Explanation: The UDDI node must be initialized (and in deactivated state) before it can be activated.

User Response: If the UDDI node is ready to be initialized, invoke the initialization operation.

CWUDM0026E: Unable to deactivate UDDI node.

Explanation: An error occurred trying to deactivate the UDDI node.

User Response: Check the running state of the UDDI application. Restart the application if necessary. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0027I: Unable to deactivate UDDI node that is not initialized.

Explanation: The UDDI node must be initialized (and activated) before it can be deactivated.

User Response: If the UDDI node is ready to be initialized, invoke the initialization operation.

CWUDM0028E: Unable to initialize UDDI node.

Explanation: An error occurred during UDDI node initialization.

User Response: Check the UDDI application installation and datasource settings. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0029I: Unable to initialize UDDI node because a required property is missing or invalid: <property name>.

Explanation: A configuration property that is required for UDDI node initialization has not been populated.

User Response: Supply a value for the required property.

CWUDM0030I: Initialize operation did not occur because the UDDI node is already initialized.

Explanation: A UDDI node can only be initialized once.

User Response: None.

CWUDM0031I: Initialize operation did not occur because the UDDI node is in default configuration and is already initialized.

Explanation: A UDDI node can only be initialized once.

User Response: None.

CWUDM0032I: The initialize operation is already in progress.

Explanation: A UDDI node can only be initialized once.

User Response: Wait until the current initialization operation completes.

CWUDM0050I: The UDDI publisher with user name <user ID> does not exist.

Explanation: An operation that requires a UDDI publisher ID could not find a publisher with that ID.

User Response: Check the UDDI publisher is registered with the UDDI node using the administrative functions available.

CWUDM0051E: Unable to create UDDI publisher with user name <user ID>.

Explanation: An error occurred trying to register the UDDI publisher.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0052I: Unable to create UDDI publisher for user name <user ID> because a UDDI publisher with that name already exists.

Explanation: The UDDI publisher is already registered.

User Response: None.

CWUDM0053E: Unable to delete UDDI publisher with user name <user ID>.

Explanation: An error occurred trying to register the UDDI publisher.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0054I: Unable to delete UDDI publisher with user name <user ID> because that UDDI publisher does not exist.

Explanation: The UDDI publisher is not registered so cannot be removed.

User Response: None.

CWUDM0055E: Unable to create a UDDI publisher with user name <user ID> because one or more entitlement identifiers are invalid.

Explanation: UddiUser objects must contain Entitlement objects with valid entitlement IDs.

User Response: Use valid entitlement IDs for Entitlement objects. Valid IDs can be found in the EntitlementConstants interface.

CWUDM0056E: Unable to update UDDI publisher with user name <user ID>.

Explanation: An error occurred trying to update the UDDI publisher.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0057E: Unable to update UDDI publisher with user name <user ID> because one or more entitlement identifiers are invalid.

Explanation: UddiUser objects must contain Entitlement objects with valid entitlement IDs.

User Response: Use valid entitlement IDs for Entitlement objects. Valid IDs can be found in the EntitlementConstants interface.

CWUDM0058I: Unable to update the UDDI publisher with user name <user ID> because that UDDI publisher does not exist.

Explanation: The UDDI publisher is not registered so cannot be updated.

User Response: None.

CWUDM0059E: Unable to retrieve information for UDDI publisher with user name <user ID>.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0060E: Unable to retrieve collection of UDDI publishers.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0061E: Unable to get tier assigned to UDDI publisher with user name <user ID>.

Explanation: This may be because the UDDI publisher with the specified user name does not exist, or because a database error occurred performing the operation.

User Response: Check the UDDI publisher exists. If it does, check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0062E: Unable to create UDDI publishers with user names: <user IDs>.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0070E: Unable to create tier with ID: <tier ID>.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0071E: Unable to delete tier with ID: <tier ID>.

Explanation: This may because the tier with the specified ID does not exist, or because a database error occurred performing the operation.

User Response: Check the tier ID. If it does exist, check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0072E: Unable to retrieve information for tier with ID: <tier ID>.

Explanation: This may be because the supplied tier ID is not an integer, or the tier with the ID does not exist, or because of a database error performing the operation.

User Response: Check the tier ID is an integer and the tier exists. If it does exist, check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0073E: Unable to retrieve collection of tiers.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0074E: Unable to set default tier to tier ID: <tier ID>.

Explanation: This may be because the tier with the specified ID does not exist, or because a database error occurred performing the operation.

User Response: Check the tier ID. If it does exist, check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0075E: Unable to update tier with ID: <tier ID>.

Explanation: This is most likely because the tier with the specified ID does not exist, or because a database error occurred performing the operation.

User Response: Check the tier ID. If it does exist, check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0076E: Unable to get count of UDDI publishers for tier ID: <tier ID>.

Explanation: This may be because the tier with the specified ID does not exist, or because a database error occurred performing the operation.

User Response: Check the tier ID. If it does exist, check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0077E: Unable to retrieve collection of entitlements.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0078E: Unable to retrieve collection of limits.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0079I: The default tier cannot be deleted.

Explanation: A tier must always exist that is designated as the tier assigned to UDDI publishers when auto-registration of UDDI publishers is enabled.

User Response: None.

CWUDM0080I: Unable to delete tier <tier ID> because it is currently assigned to a UDDI publisher.

Explanation: Tiers which have UDDI publishers assigned to them cannot be deleted.

User Response: If you want to remove the tier, assign the UDDI publishers to a different tier first.

CWUDM0100E: Unable to get configuration property information for property with ID: <property ID> **Explanation:** A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0101I: Unable to get configuration property information for property with ID: <property ID> because it does not exist.

Explanation: The configuration property with the specified ID does not exist.

User Response: None.

CWUDM0102E: Unable to retrieve collection of configuration properties.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0103E: Unable to update configuration properties.

Explanation: This may occur if any of the updated property objects fails validation. Check any cause exceptions for possible additional information. Alternatively database error may have occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0104E: Unable to update configuration property with ID: <property ID>

Explanation: This may occur if the updated property object fails validation. Check any cause exceptions for possible additional information. Alternatively a database error may have occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0105I: Unable to update configuration property with ID: <property ID> because it does not exist. **Explanation:** The configuration property with the specified ID does not exist.

User Response: Use a valid configuration property ID. These can be found in PropertyConstants.

CWUDM0106I: Unable to update configuration properties because one or more properties do not exist in the UDDI node.

Explanation: One or more supplied configuration properties do not exist in the UDDI registry.

User Response: Use valid configuration property IDs. These can be found in PropertyConstants.

CWUDM0107E: Failed to retrieve required properties from database.

Explanation: This indicates a database error occurred when trying to initialize a UDDI node.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0120E: Unable to get policy information for policy with ID: <policy ID>.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0121I: Unable to get policy information for policy with ID: <policy ID> because it does not exist. **Explanation:** The policy with the specified ID does not exist.

User Response: None.

CWUDM0122E: Unable to get policy group with group ID: <policy group ID>.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0123E: Unable to retrieve collection of policy groups.

Explanation: A database error occurred performing the operation.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0124E: Unable to update policies.

Explanation: This may occur if any of the updated policy objects fails validation. Check any cause exceptions for possible additional information. Alternatively a database error may have occurred performing the operation.

User Response: Ensure all policy IDs and values are valid. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0125E: Unable to update policy with ID: <policy ID>.

Explanation: This may occur if the updated policy object fails validation or the ID isn't recognized. Check any cause exceptions for possible additional information. Alternatively a database error may have occurred performing the operation.

User Response: Check the policy ID and value are valid. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0126I: Unable to get policy group information for policy group with ID: <policy group ID> because it does not exist.

Explanation: The policy group with the specified ID does not exist.

User Response: Use a valid policy group ID. These can be found in PolicyConstants.

CWUDM0127I: Unable to update policies because one or more policies do not exist in the UDDI node. **Explanation:** One or more supplied policies do not exist in the UDDI registry.

User Response: Use valid policy IDs. These can be found in PolicyConstants.

CWUDM0128I: Unable to update policy with ID: <policy ID> because it does not exist.

Explanation: The policy with the specified ID does not exist.

User Response: Use a valid policy ID. These can be found in PolicyConstants.

CWUDM0140E: Unable to change value set tModelKey from <original tModel key> to <new tModel key>. **Explanation:** The original tModel key or new tModel key supplied could not be found in the UDDI registry, or possibly there was a database error.

User Response: Supply keys for tModels that exist in the UDDI registry. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0141E: Unable to retrieve value set details for tModel key: <tModel key>.

Explanation: A database error occurred getting the value set details.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0142E: Unable to get value set property: <value set property ID> for value set with tModel key: <tModel key>.

Explanation: The value set property ID or tModel key supplied was not recognized.

User Response: Supply a valid value set property ID and tModel key. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0143E: Unable to retrieve value sets collection.

Explanation: A database error may have occurred.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0144E: Unable to determine if value set with tModel key: <tModel key> exists.

Explanation: The tModel key supplied is not valid or a database error may have occurred.

User Response: Ensure the tModel exists. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0145E: Unable to load value set data for value set with tModel key: <tModel key> and file name <1>.

Explanation: The original tModel key or new tModel key supplied could not be found in the UDDI registry, or possibly there was a database error.

User Response: Ensure the tModel exists and the value set data object is populated. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0146E: Unable to load value set data for value set with tModel key: <tModel key>.

Explanation: The tModel key is not recognized or a database error occurred.

User Response: Ensure the tModel exists and the value set data object is populated. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0147E: Could not update value set status with tModel key: <tModel key>.

Explanation: The value set status object supplied is null or the tModel key is not known.

User Response: Supply a valid value set status object with tModel key for a value set tModel that exists. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0148E: Could not update value set status with tModel key: <tModel key>, property: <value set property ID>.

Explanation: A database error occurred updating the supported status of the value set status.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0149E: Could not update value set status with tModel key: <tModel key>.

Explanation: One of the supplied value set status objects contained a tModel key which was not recognized.

User Response: Supply tModel keys for value set tModels that exist. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0150E: Could not update value set status with tModel key: <tModel key>, property: <value set property ID>.

Explanation: A database error occurred updating the supported status of one of the value set status objects.

User Response: Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0151E: Unable to unload value set data with tModel key: <tModel key>.

Explanation: The tModel key is not recognized or a database error occurred.

User Response: Ensure the tModel exists. Check UDDI application configuration and database connectivity. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0170E: Loading of configuration file <configuration file URL> failed.

Explanation:

User Response: Check UDDI application configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0171E: Parsing of configuration file <configuration file URL> failed.

Explanation:

User Response: Check UDDI application configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0172W: An unexpected date format was found while parsing configuration file.

Explanation:

User Response: Check UDDI application configuration. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDM0180I: UDDI node was activated.

Explanation: The UDDI node is ready to accept UDDI API requests.

User Response: None.

CWUDM0181I: UDDI node was deactivated.

Explanation: The UDDI node is set to not accept UDDI API requests. This typically occurs when configuration settings are being updated by administrative tasks.

User Response: None.

CWUDM0182I: UDDI node initialized successfully.

Explanation: The UDDI node is ready to accept UDDI API requests.

User Response: None.

CWUDM0183I: UDDI publisher <user ID> was created.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0184I: UDDI publisher <user ID> was updated.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0185I: UDDI publisher <user ID> was deleted.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0186I: Tier <tier ID> was created.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0187I: Tier <tier ID> was updated.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0188I: Tier <tier ID> was deleted.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0189I: Configuration property <property ID> was updated to value <property value>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0190I: Policy <policy ID> was updated to value <policy value>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0191I: Default tier was set to <tier ID>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0192I: Loaded value set for tModel key <tModel key> from file <1>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0193I: Loaded value set for tModel key <tModel key>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0194I: Unloaded value set for tModel key <tModel key>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0195I: Updated value set supported status for tModel key <tModel key> to <supported status>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0196I: Changed tModel key for value set from <original tModel key> to <new tModel key>.

Explanation: This message indicates the administrative operation completed successfully.

User Response: None.

CWUDM0220W: <value or ID> is too short. The length must be in the range <minimum length> to <maximum length> characters.

Explanation: A value or ID of type String is shorter than expected.

User Response: Supply a value or ID that is in the expected range.

CWUDM0221W: <value or ID> is too long. The length must be in the range <minimum length> to <maximum length> characters.

Explanation: A value or ID of type String is longer than expected.

User Response: Supply a value or ID that is in the expected range.

CWUDM0222W: <value or ID> is read-only and cannot be updated.

Explanation: The configuration setting is marked as read only, which may be because the setting is for information only, or because the setting is configured once during UDDI node initialization, such as the UDDI node ID or root key generator.

User Response: None.

CWUDM0223W: <value or ID> is required.

Explanation: The configuration setting expects a value to be entered.

User Response: Supply a non-empty, valid value for the configuration setting.

CWUDM0224W: <value or ID> must be a valid URL value.

Explanation: The configuration property is expected to be a valid URL.

User Response: Supply a valid URL.

CWUDM0225W: <value or ID> must be a valid xml:lang value.

Explanation: The configuration property supplied is not a valid xml:lang value.

User Response: Supply a valid xml:lang value.

CWUDM0226W: <value or ID> must be of type: <value type>.

Explanation: The value or ID is not of the expected type.

User Response: Supply a value or ID of the correct type.

CWUDM0227W: <value or ID> cannot be a null value.

Explanation: A property ID was expected but a null value was supplied.

User Response: Supply a valid, non-null ID.

CWUDM0228W: <value or ID> must be in the range <minimum value> to <maximum value>.

Explanation: The value is not in the expected range

User Response: Supply a value in the expected range.

CWUDM0229W: The entitlement object with ID: <entitlement ID> is not valid.

Explanation: The UDDI application doesn't recognize the entitlement with the supplied ID.

User Response: Supply a valid entitlement ID, which can be found in EntitlementConstants.

CWUDM0230W: The limit object with ID: <limit ID> is not valid.

Explanation: The UDDI application doesn't recognize the limit with the supplied ID.

User Response: Supply a valid limit ID, which can be found in LimitConstants.

CWUDM0231W: <value> must be a valid UDDI key value.

Explanation: The value supplied was not a UDDI key.

User Response: UDDI keys must start with the text 'uddi:'. See the UDDI specification for further information about UDDI keys.

CWUDM0232W: <value> must be a valid UDDI key generator key.

Explanation: The value supplied was not a UDDI key generator.

User Response: UDDI key generator values must be valid UDDI keys and end in the string 'keygenerator'. See the UDDI specification for further information about UDDI keys.

CWUDM0240W: The configuration property ID cannot be null or empty.

Explanation: The ID supplied was null or empty.

User Response: Supply a non-null, non-empty ID, valid for the context in which it is used.

CWUDM0241W: The policy group ID cannot be null or empty.

Explanation: The ID supplied was null or empty.

User Response: Supply a non-null, non-empty ID, valid for the context in which it is used.

CWUDM0242W: The policy ID cannot be null or empty.

Explanation: The ID supplied was null or empty.

User Response: Supply a non-null, non-empty ID, valid for the context in which it is used.

CWUDM0243W: The entitlement ID cannot be null or empty.

Explanation: The ID supplied was null or empty.

User Response: Supply a non-null, non-empty ID, valid for the context in which it is used.

CWUDM0244W: The limit ID cannot be null or empty.

Explanation: The ID supplied was null or empty.

User Response: Supply a non-null, non-empty ID, valid for the context in which it is used.

CWUDM0245W: The user ID cannot be null or empty.

Explanation: The ID supplied was null or empty.

User Response: Supply a non-null, non-empty ID, valid for the context in which it is used.

CWUDM0246W: The tier ID cannot be null or empty.

Explanation: The ID supplied was null or empty.

User Response: Supply a non-null, non-empty ID, valid for the context in which it is used.

CWUDM0250W: The configuration property parameter cannot be null or empty.
Explanation: The parameter supplied was null or empty.

User Response: Supply a non-null, non-empty configuration property parameter.
CWUDM0251W: The policy group parameter cannot be null or empty.
Explanation: The parameter supplied was null or empty.

User Response: Supply a non-null, non-empty policy group parameter.
CWUDM0252W: The policy parameter cannot be null or empty.
Explanation: The parameter supplied was null or empty.

User Response: Supply a non-null, non-empty policy parameter.
CWUDM0253W: The entitlement parameter cannot be null or empty.
Explanation: The parameter supplied was null or empty.

User Response: Supply a non-null, non-empty entitlement parameter.
CWUDM0254W: The limit parameter cannot be null or empty.
Explanation: The parameter supplied was null or empty.

User Response: Supply a non-null, non-empty limit parameter.
CWUDM0255W: The user parameter cannot be null or empty.
Explanation: The parameter supplied was null or empty.

User Response: Supply a non-null, non-empty user (UDDI publisher) parameter.
CWUDM0256W: The tier parameter cannot be null or empty.
Explanation: The parameter supplied was null or empty.

User Response: Supply a non-null, non-empty tier parameter.
CWUDM0257I: The collection cannot be null.
Explanation: The collection (typically a list of properties, policies, tiers or users to update) parameter supplied was null.

User Response: Supply a non-null collection parameter.

CWUDNnnnns (Web Services UDDI Node Manager) messages: Most of these messages are in the format 'Error', Class, Method, Exception. For example

```
CWUDN0005E: Error, NodeManager, txnlInit, UDDIFatalErrorException during init
```

indicates that a UDDIFatalErrorException was thrown by the txnlInit method of the NodeManager class, during execution of the init method.
CWUDN0001I: UDDI Node State change, new state:
Explanation: The UDDI has successfully changed to the identified new state.

User Response: None.
CWUDN0002I: Error, invalid Node State requested:
Explanation: The UDDI Registry detected a request for invalid state changes.

User Response: Contact the IBM Customer Service Center.
CWUDN0004E: Error, NodeManager, txnlInit, failed getting persister control
Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.
CWUDN0005E: Error, NodeManager, txnlInit, UDDIFatalErrorException during init
Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.
CWUDN0006E: Error, NodeManager, txnlInit, UDDIException during init.
Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.
CWUDN0007E: Error, NodeManager, txnlInit, Exception during init
Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0008E: Error, NodeManager, txnIInit, Throwable during init

Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0009E: Error, NodeManager, txnIInit, rollback exception

Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0010E: Error, NodeManager, txnIInit, Throwable releasing connection

Explanation: NodeManager initialization failure.

User Response: Contact the IBM Customer Service Center.

CWUDN0011E: Error, NodeManager, txnIDestroy, failed getting Persister control

Explanation: NodeManager application stop or removal error.

User Response: Contact the IBM Customer Service Center.

CWUDN0012E: Error, NodeManager, txnIDestroy, UDDI Exception during destroy

Explanation: NodeManager application stop or removal error.

User Response: Contact the IBM Customer Service Center.

CWUDN0013E: Error, NodeManager, txnIDestroy, rollback Exception

Explanation: NodeManager application stop or removal error.

User Response: Contact the IBM Customer Service Center.

CWUDN0014E: Error, NodeManager, txnIDestroy, Exception releasing connection

Explanation: NodeManager application stop or removal error.

User Response: Contact the IBM Customer Service Center.

CWUDQnnnns (Web Services UDDI Migration) messages:

CWUDQ0001I: UDDI registry migration datasource is present.

Explanation: The UDDI migration datasource is defined when UDDI is started.

User Response: None.

CWUDQ0002I: UDDI registry migration has started.

Explanation: The UDDI database migration process has started. This may take some time to complete dependent on the size of your database.

User Response: None.

CWUDQ0003I: UDDI registry migration has completed.

Explanation: The UDDI database migration process has finished.

User Response: None.

CWUDQ0004W: UDDI registry not started due to migration errors.

Explanation: The UDDI registry has not been activated because migration errors were encountered.

User Response: Examine the messages produced during the migration process. Determine if the problem is user fixable, that is the Version 2 database is quiesced and so on. If the problem is fixable recreate the Version 3 UDDI database and restart the UDDI application. If the problem cannot be rectified contact your IBM Customer Service Center. It is still possible to use the UDDI Administration Console to start the UDDI registry.

CWUDQ0005I: <rows> rows have been inserted into table <table>.

Explanation: An informational message showing the number of database rows converted for each UDDI table.

User Response: None.

CWUDQ10001E: Row not inserted into <table>.

Explanation: A row was unable to be inserted into a database table.

User Response: See CWUDQ1003E for details of the SQL Exception details

CWUDQ1002E: Table <table> values are: <Key Values>.

Explanation: Contains the table in error and the significant key values of the row being migrated.

User Response: None.

CWUDQ1003E: SQL Exception during migration: <SQL Exception Message>.

Explanation: An SQL Exception has occurred during the migration process. The SQL Exception details are displayed in the message.

User Response: Examine the SQL Exception information on its cause. Message CWUDQ1002E contains table details and significant key values. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDQ1004E: <rows> rows have not been inserted into table <table>.

Explanation: A message showing the number of database rows not converted for a particular UDDI table.

User Response: None.

CWUDQ1005E: SQL Exception during key value extraction: <SQL Exception Message>.

Explanation: An SQL Exception occurred during the production of the CWUDQ1002E message.

User Response: None.

CWUDQ1006E: Exception during migration: <Exception Message>.

Explanation: An Exception has occurred during the migration process. The Exception details are displayed in the message.

User Response: Examine the Exception information on its cause. Message CWUDQ1002E contains table details and significant key values. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDRnnnns (Web Services UDDI Logging and Tracing) messages:

CWUDR0001E: Exception "<exception>" occurred while attempting to get UDDI Message Logger.

Explanation: This message is issued to stderr when an attempt to get the UDDI Message Logger fails with the indicated exception. Since the attempt to get the message logger failed, the message cannot be logged. No messages can be logged by this instance of the IBM WebSphere UDDI Registry.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then contact the IBM Customer Service Center.

CWUDR0002E: Exception "<exception>" occurred while attempting to get UDDI Trace Logger for "<component>".

Explanation: This message is logged when an attempt to get the UDDI Trace Logger for the specified component (or package) fails with the indicated exception. No trace entries can be logged for this component or package of the IBM WebSphere UDDI Registry.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then contact the IBM Customer Service Center.

CWUDSnnnns (Web Services UDDI SOAP Interface) messages:

CWUDS0001E: ParserPool found empty whilst attempting to process request. Request unsatisfied

Explanation: A SOAP request was received, but was unable to be dealt with, as there were no free Parsers within the ParserPool.

User Response: Consider increasing the number of Parsers within the ParserPool by modifying the Init Parameter on the SOAP servlets.

CWUDS0002E: Error locating schemas required for UDDI processing. SOAP Servlets unworkable.

Explanation: The SOAP servlet was unable to locate the schemas it requires in order to process SOAP requests. Without these, the servlet cannot process SOAP requests.

User Response: Check installation of UDDI was performed correctly. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then contact the IBM Customer Service Center.

CWUDS0003W: Servlet unable to locate init parameter 'defaultPoolSize'. Using internal defaults.

Explanation: The SOAP servlet was unable to locate the init parameter which sets the default size of the ParserPool. It will fall back to an internal default.

User Response: If this message occurred after attempting to make changes to the defaultPoolSize init parameter, ensure the changes were correct. If this message has appeared after installed, ensure installation was performed correctly.

CWUDS0004W: Servlet unable to understand init parameter 'defaultPoolSize'. Using internal defaults.

Explanation: The SOAP servlet was unable to parse the init parameter which sets the default size of the ParserPool. It will fall back to an internal default.

User Response: If this message occurred after attempting to make changes to the defaultPoolSize init parameter, ensure the changes were correct. If this message has appeared after installed, ensure installation was performed correctly.

CWUDS0005E: Error occurred during parser creation.

Explanation: An unspecified error occurred during the creation of a SOAP parser

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then contact the IBM Customer Service Center.

CWUDS0006E: Internal configuration error.

Explanation: This error may occur if there was a failure creating a Parser, with accompanying message CWUDS0005. It may also occur if there was a problem acquiring the Persistence layer.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then please contact the IBM Customer Service Center.

CWUDS0007E: Error during servlet acquisition of persistence layer.

Explanation: The SOAP servlet was unable to acquire the persistence layer required for it to communicate with the UDDI datasource

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then contact the IBM Customer Service Center.

CWUDS0008E: Error during servlet release of persistence layer.

Explanation: The persistence layer reported a problem when the SOAP servlet attempted to release it.

User Response: Restart the UDDI registry. If the error persists, examine the WebSphere logs for information on its cause. If the problem cannot be resolved, then contact the IBM Customer Service Center.

CWUDS0009E: Error during sending of response to client.

Explanation: An error occurred when sending a SOAP response message back to a client. The client may not have received the response

User Response: This error is recorded to enable logging of failed responses to clients. The error may be the fault of the client disconnecting before the reply could be sent, or may indicate a network problem. Examine the WebSphere logs for more information on its cause.

CWUDS0010E: Serious error while servlet attempting to process doPost Request.

Explanation: A serious error occurred processing a UDDI Version 1 or Version 2 SOAP request.

User Response: Check the validity of the UDDI Version 1 or Version 2 SOAP request.

CWUDS0011E: incomplete URL received processing doGet Request.

Explanation: An incomplete or incorrect UDDI Version 3 GET request was received.

User Response: Check the validity of the UDDI Version 3 GET request URL.

CWUDS0012E: non-UDDI exception received processing doGet Request.

Explanation: A serious error occurred while processing a UDDI Version 3 GET request.

User Response: Check the validity of the UDDI Version 3 GET request URL.

CWUDTnnnns (Web Services UDDI Registry Transaction Manager) messages:

CWUDT0001E: UDDIFatalErrorException processing request. persistenceManager.getControl() returned null, throwing UDDIFatalErrorException.

Explanation: A serious internal error occurred while processing a UDDI Version 3 SOAP request.

User Response: Check the configuration of the UDDI Enterprise Application. If the problem cannot be resolved, contact the IBM Customer Service Center.

CWUDT0002E: UDDIFatalErrorException processing request. UDDIFatalErrorException processing request.

Explanation: A serious UDDI error occurred while processing a UDDI Version 3 SOAP request.

User Response: Check the validity of the UDDI request.

CWUDT0003E: UDDIFatalErrorException processing request. Exception processing request.

Explanation: A serious non UDDI error occurred while processing a UDDI Version 3 SOAP request.

User Response: Check the validity of the UDDI request.

CWUDUnnnns (Web Services UDDI Utility Tools) messages:

```
CWUDU0001I: 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 /uddikeys.txt
```

Explanation: This is the usage message displayed at the command line when the user has entered an invalid combination of arguments or options.

User Response: Enter the command according to the usage message.

CWUDU0002I: ******** Starting UDDI Utility Tools ************

Explanation: This message is used as a marker in the message log file to indicate tool start points.

User Response: None.

CWUDU0003I: Promoting entityType<entity type> key<entity key>...

Explanation: Indicates which entity type (business, tModel and so on) is being promoted, and it's key value.

User Response: None.

CWUDU0004I: Bad entityType: received<incorrect entity type>, expected <tModel|business|service|binding>

Explanation: The user entered an incorrect entity type.

User Response: Use an entity type of tModel, business, service or binding.

CWUDU0005I: Promotion successful.

Explanation: Indicates the promote function completed successfully.

User Response: None.

CWUDU0006I: Import successful.

Explanation: Indicates the import function completed successfully.

User Response: None.

CWUDU0007I: Export successful.

Explanation: Indicates the export function completed successfully.

User Response: None.

CWUDU0008I: Delete successful.

Explanation: Indicates the delete function completed successfully.

User Response: None.

CWUDU0009I: Exporting entities ...

Explanation: Indicates the export function has started.

User Response: None.

CWUDU0010I: Exporting business, businessKey[<business key].

Explanation: Indicates that the businessEntity with the specified key is being exported.

User Response: None.

CWUDU0011I: Exporting service, serviceKey[<service key>].

Explanation: Indicates that the businessService with the specified key is being exported.

User Response: None.

CWUDU0012I: Exporting binding, bindingKey[<binding key>].

Explanation: Indicates that the bindingTemplate with the specified key is being exported.

User Response: None.

CWUDU0013I: Exporting tModel, tModelKey[<tModel key>].

Explanation: Indicates that the tModel with the specified key is being exported.

User Response: None.

CWUDU0014I: Exporting referenced tModel, tModelKey[<tModel key>].

Explanation: Indicates that the referenced tModel with the specified key is being exported.

User Response: None.

CWUDU0015I: Exported <entity count> entities.

Explanation: Indicates that the export function completed, and shows the number of entities exported.

User Response: None.

CWUDU0016I: Importing entities ...

Explanation: Indicates the import function has started.

User Response: None.

CWUDU0017I: Importing business, businessKey[<business key>].

Explanation: Indicates that the businessEntity with the specified key is being imported.

User Response: None.

CWUDU0018I: Importing service, serviceKey[<service key>].

Explanation: Indicates that the businessService with the specified key is being imported.

User Response: None.

CWUDU0019I: Importing binding, bindingKey[<binding key>]

Explanation: Indicates that the bindingTemplate with the specified key is being imported.

User Response: None.

CWUDU0020I: Importing tModel, tModelKey[<tModel key>].

Explanation: Indicates that the tModel with the specified key is being imported.

User Response: None.

CWUDU0021I: Importing referenced tModel, tModelKey[<tModel key>].

Explanation: Indicates that the referenced tModel with the specified key is being imported.

User Response: None.

CWUDU0022I: Imported <entity count> entities.

Explanation: Indicates that the import function completed, and shows the number of entities imported.

User Response: None.

CWUDU0023I: Deleting entities ...

Explanation: Indicates the delete function has started.

User Response: None.

CWUDU0024I: Deleting business, businessKey[<business key>].

Explanation: Indicates that the businessEntity with the specified key is being deleted.

User Response: None.

CWUDU0025I: Deleting service, serviceKey[<service key>].

Explanation: Indicates that the businessService with the specified key is being deleted.

User Response: None.

CWUDU0026I: Deleting binding, bindingKey[<binding key>].

Explanation: Indicates that the bindingTemplate with the specified key is being deleted.

User Response: None.

CWUDU0027I: Deleting tModel, tModelKey[<tModel key>].

Explanation: Indicates that the tModel with the specified key is being deleted.

User Response: None.

CWUDU0028I: Deleted <entity count> entities.

Explanation: Indicates that the delete function completed, and shows the number of entities deleted.

User Response: None.

CWUDU0029I: Serializing ...

Explanation: Indicates that generation of the Entity Definition File has started.

User Response: None.

CWUDU0030I: Serialized entities.

Explanation: Indicates that generation of the Entity Definition File completed successfully.

User Response: None.

CWUDU0031I: Deserializing ...

Explanation: Indicates that reading of the Entity Definition File and creation of UDDI entities has started.

User Response: None.

CWUDU0032I: Deserialized entities.

Explanation: Indicates that reading of the Entity Definition File and creation of UDDI entities completed successfully.

User Response: None.

CWUDU0033I: Function '<function>' completed successfully.

Explanation: Indicates the requested function completed successfully.

User Response: None.

CWUDU0034W: Function '<function>' did not complete successfully. See messages log for further information.

Explanation: Indicates the requested function did not complete successfully.

User Response: The messages log may yield further information if the verbose option is on. Check the configuration properties file setting are correct. If that does not identify the problem, try running with trace logging enabled. If that does not yield a solution, contact your IBM support center.

CWUDU0035W: Parser error: <warning description>

Explanation: The XML parser reports a warning about the content of the Entity Definition File.

User Response: Based on the context of the warning message, check the validity of the Entity Definition File.

CWUDU0036E: Parser error <error description>

Explanation: The XML parser reports an error about the content of the Entity Definition File.

User Response: Based on the context of the error message, check the validity of the Entity Definition File.

CWUDU0037E: Unrecognized parser feature: <feature description>

Explanation: A parser feature set by the UDDI Utility Tools is not recognized by the parser.

User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.

CWUDU0038E: Unsupported parser feature: <feature description>

Explanation: A parser feature set by the UDDI Utility Tools is not supported by the parser.

User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.

CWUDU0039E: Unrecognized parser property: <property description>, value: <value>

Explanation: A parser property set by the UDDI Utility Tools is not recognized by the parser.

User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.

CWUDU0040E: Unsupported parser property: <property description>, value: <value>

Explanation: A parser property set by the UDDI Utility Tools is not supported by the parser.

User Response: Check you are using the correct type and level of XML parser. If correct, contact your IBM support center.

CWUDU0042E: Unable to find the configuration file: <filepath>

Explanation: UDDI Utility Tools cannot locate the specified configuration file.

User Response: UDDI Utility Tools looks for a default configuration properties with the file name 'UDDIUtilityTools.properties' in the current directory. Check that the configuration file has this name, or that the argument value supplied with the '-properties' option is pointing at a file that exists.

CWUDU0043E: An Exception occurred trying to read the configuration file.

Explanation: The configuration file could not be read.

User Response: Check the file path points to a valid file and that the current user has permission to read the file.

CWUDU0044W: Configuration file is missing the '<property name>' property.

Explanation: A required property is missing from the configuration file.

User Response: Add the missing property name and value to the configuration file. Check that the property name is not misspelled.

CWUDU0045W: Property: '<property name>' has value '<property value>'. It must be either 'true' or 'false'.

Explanation: A value was given to a property other than 'true' or 'false'.

User Response: Set the property value to 'true' or 'false'.

CWUDU0046W: Property: '<property name>' has value '<property value>'. It must be an integer value. **Explanation:** A value was given to a property other than an integer value.

User Response: Set the property value to an integer value.

CWUDU0047E: Unable to find the keyFile file: <keys file path>

Explanation: The keys file could not be located at the specified path.

User Response: Check the file name and path and correct and that the file exists.

CWUDU0048E: Unable to read the keyFile file: <keys file path>

Explanation: The keys file could not be read due to an IO error.

User Response: Check that the current user has permission to read the file.

CWUDU0049E: Unable to write to entity definition file: <entity definition file path>

Explanation: During serialization, the Entity Definition File could not be written to.

User Response: Check the file's read only attribute is not set.

CWUDU0050E: Unable to find UDDI Entity definition file: <entity definition file path>

Explanation: The Entity Definition File could not be found at the specified file path.

User Response: Check the file path is correct and that the file exists.

CWUDU0051E: Unable to read UDDI Entity definition file: <entity definition file path>

Explanation: The Entity Definition File could not be read due to an IO error.

User Response: Check that the current user has permission to read the file.

CWUDU0052E: Unable to close the message file: <file path>

Explanation: The attempt to close the message log file failed.

User Response: The disk might be full. If so, clear some space or direct log output to a different disk.

CWUDU0053E: Unable to close the trace file: <file path>

Explanation: The attempt to close the trace log file failed.

User Response: The disk might be full. If so, clear some space or direct log output to a different disk.

CWUDU0054E: The logger was unable to find the file: <file path>

Explanation: The UDDI Utility Tools logger could not find the specified file.

User Response: None.

CWUDU0055E: ERROR OCCURRED ...

Explanation: General purpose error message used in development only.

User Response: None.

CWUDU0056E: Exception:

Explanation: General purpose message prefix used for reporting exceptions.

User Response: None.

CWUDU0057W: Only one function may be specified on the command line.

Explanation: Multiple function commands were entered on the command line.

User Response: Specify one function in accordance with the usage message.

CWUDU0058W: No function was specified.

Explanation: UDDI Utility Tools was invoked with no function specified.

User Response: Specify one function in accordance with the usage message.

CWUDU0059W: The function: <function> was not recognized.

Explanation: The function value did not match any of the allowed functions.

User Response: Specify one function in accordance with the usage message.

CWUDU0060W: The argument '<argument>' was not recognized.

Explanation: The argument value does not match any of the allowed arguments.

User Response: Specify arguments in accordance with the usage message.

CWUDU0061W: There was a missing value for <argument> argument.

Explanation: An expected value for the specified argument was not supplied.

User Response: Specify a value for the argument in accordance with the usage message.

CWUDU0062W: Unexpected argument: <argument> (entity key file is already specified).

Explanation: The entity type argument cannot be specified if the keysFile argument has already been specified.

User Response: Specify arguments in accordance with the usage message.

CWUDU0063W: Unexpected argument: <argument> (entity key is already specified).

Explanation: The keysFile argument cannot be specified if an entity type argument and key value has already been specified.

User Response: Specify arguments in accordance with the usage message.

CWUDU0064W: Argument: <argument> cannot be specified more than once.

Explanation: An argument was specified twice in the same command.

User Response: Specify arguments in accordance with the usage message.

CWUDU0065E: No entity keys were specified.

Explanation: A keys file or an entity type and key value must be specified for functions using keys.

User Response: Specify arguments in accordance with the usage message.

CWUDU0066E: Could not load Database driver: dbDriver<database driver>.

Explanation: The specified database driver could not be loaded.

User Response: Check the database driver value in the configuration file is valid, and the driver's class is present in the classpath property.

CWUDU0067E: Could not create Database connection: dbUrl, dbUser, (dbPasswd not shown).

Explanation: A connection could not be established with the database at the specified URL with the specified userid.

User Response: Check the database URL, userid and password values are correct n the configuration file, and that the database manager is running.

CWUDU0068E: Could not close the database connection.

Explanation: An attempt to close the database connection failed.

User Response: If the problem persists, contact your IBM support center.

CWUDU0069E: Could not create minimal entity for tModel.

Explanation: The minimal data necessary for a valid tModel could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

CWUDU0070E: Could not create minimal entity for Service.

Explanation: The minimal data necessary for a valid businessService could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

CWUDU0071E: Could not create minimal entity for Business.

Explanation: The minimal data necessary for a valid businessEntity could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

CWUDU0072E: Could not create minimal entity for Binding.

Explanation: The minimal data necessary for a valid bindingTemplate could not be inserted in the target UDDI registry database.

User Response: Check the database URL, userid and password values are correct in the configuration file, and that the database manager is running.

CWUDU0073E: There was an error while trying to create an XML Document.

Explanation: An attempt to create the Entity Definition File failed.

User Response: Check the file path specified in the configuration file for the Entity Definition File is valid and is not set to be read only.

CWUDU0074E: There was an error parsing the entity definition file.

Explanation: An unspecified error occurred when parsing the Entity Definition File.

User Response: Check the entity definition file content is valid according to the UDDI Utility Tools schema file, promoter.xsd.

CWUDU0075E: One or more errors occurred while parsing the entity definition file. See message log for details.

Explanation: Errors occurred when parsing the Entity Definition File.

User Response: Check the entity definition file content is valid according to the UDDI Utility Tools schema file, promoter.xsd.

CWUDU0076W: One or more warnings were raised while parsing the entity definition file. See message log for details.

Explanation: Warnings occurred when parsing the Entity Definition File.

User Response: Check the entity definition file content is valid according to the UDDI Utility Tools schema file, promoter.xsd.

CWUDU0078E: Unable to obtain authinfo.

Explanation: AuthInfo could not be obtained from the UDDI registry with the given userid and password.

User Response: Check the userid and password property values are correct in the configuration file.

CWUDU0079E: The inquiryURL is malformed: <inquiry URL>.

Explanation: The inquiry URL specified in the configuration file is not valid.

User Response: Correct the value for the inquiry URLs (fromInquiryURL and toInquiryURL) in the configuration file.

CWUDU0080E: The publishURL is malformed: <publish URL>

Explanation: The publish URL specified in the configuration file is not valid.

User Response: Correct the value for the publish URL (toPublishURL) in the configuration file.

CWUDU0081E: Could not get tModel detail for tModelKey[<tModel key>].

Explanation: The get tModel operation failed on the source registry.

User Response: Check the key exists in the source registry.

CWUDU0082E: Could not get service detail for serviceKey[<service key>].

Explanation: The get service operation failed on the source registry.

User Response: Check the key exists in the source registry.

CWUDU0083E: Could not get business detail for businessKey[<business key>].

Explanation: The get business operation failed on the source registry.

User Response: Check the key exists in the source registry.

CWUDU0084E: Could not get binding detail for bindingKey[<binding key>].

Explanation: The get binding operation failed on the source registry.

User Response: check the key exists in the source registry.

CWUDU0085E: Could not save tModel for tModelKey[<tModel key>].

Explanation: The publish operation failed at the target registry.

User Response: Check the tModel is not referencing another entity (such as a tModel) that is not present in the target registry. This may occur if the 'importReferenced' property is set to 'false'. Specify referenced tModels in the referencedtModels section of the Entity Definition File and set 'importReferenced' property in the configuration file to 'true'.

CWUDU0086E: Could not save business for businessKey[<business key>].

Explanation: The publish operation failed at the target registry.

User Response: Check the businessEntity is not referencing another entity (such as a tModel) that is not present in the target registry. This may occur if the 'importReferenced' property is set to 'false'. Specify referenced tModels in the referencedtModels section of the Entity Definition File and set 'importReferenced' property in the configuration file to 'true'.

CWUDU0087E: Could not save service for parent businessKey[<business key>].

Explanation: The publish operation failed at the target registry.

User Response: Check the businessEntity specified as the parent of the businessService exists in the target registry.

CWUDU0088E: Could not save binding for parent serviceKey[<service key>].

Explanation: The publish operation failed at the target registry.

User Response: Check the businessService specified as the parent of the bindingTemplate exists in the target registry.

CWUDU0089W: Did not save service for serviceKey[<service key>] because parent business did not exist.

Explanation: The parent business for the specified businessService does not exist.

User Response: Check the key value for the parent entity is correct in the Entity Definition File, and that the entity exists in the target registry.

CWUDU0090W: Did not save binding for bindingKey[<binding key>] because parent service did not exist. **Explanation:** The parent service for the specified bindingTemplate does not exist.

User Response: Check the key value for the parent entity is correct in the Entity Definition File, and that the entity exists in the target registry.

CWUDU0091E: Could not delete business for businessKey[<businness key>].

Explanation: The UDDI4J operation to delete the businessEntity with the specified key failed.

User Response: Check the userid and password property values in the configuration file and that the entity exists in the target UDDI registry.

CWUDU00921E: Could not delete service for serviceKey[<tModel key>].

Explanation: The UDDI4J operation to delete the businessService with the specified key failed.

User Response: Check the userid and password property values in the configuration file and that the entity exists in the target UDDI registry.

CWUDU0093E: Could not delete binding for bindingKey[<binding key>].

Explanation: The UDDI4J operation to delete the bindingTemplate with the specified key failed.

User Response: Check the userid and password property values in the configuration file and that the entity exists in the target UDDI registry.

CWUDU0094E: Could not delete tModel for tModelKey[<tModel key>].

Explanation: The UDDI4J operation to delete the tModel with the specified key failed.

User Response: Check the userid and password property values in the configuration file and that the entity exists in the target UDDI registry.

CWUDU0096W: <entity type><key value> is not a valid UUID.

Explanation: The key value entered does not comply with the format specified for a UUID in the UDDI specification.

User Response: Enter a valid UUID key.

CWUDU0097W: Did not save tModel for tModelKey[<tModel key>] as it already exists. Use the -overwrite argument to overwrite the tModel.

Explanation: The tModel was not saved because the overwrite property is false

User Response: If the desired action is to overwrite existing entities, specify -overwrite on the command line or set the overwrite property in the configuration file to true.

CWUDU0098W: Did not save business for businessKey[<business key>] as it already exists. Use the -overwrite argument to overwrite the business.

Explanation: The businessEntity was not saved because the overwrite property is false

User Response: If the desired action is to overwrite existing entities, specify -overwrite on the command line or set the overwrite property in the configuration file to true.

CWUDU0099W: Did not save service for serviceKey[<service key key>] as it already exists. Use the -overwrite argument to overwrite the service.

Explanation: The businessService was not saved because the overwrite property is false

User Response: If the desired action is to overwrite existing entities, specify -overwrite on the command line or set the overwrite property in the configuration file to true.

CWUDU0100W: Did not save binding for bindingKey[<binding key key>] as it already exists. Use the -overwrite argument to overwrite the binding.

Explanation: The bindingTemplate was not saved because the overwrite property is false

User Response: If the desired action is to overwrite existing entities, specify -overwrite on the command line or set the overwrite property in the configuration file to true.

CWUDU0101W: Bad entity type: received<entity type>, expected <tModel|business|service|binding>.

Explanation: The entered entity type was not recognized.

User Response: Specify arguments in accordance with the usage message.

CWUDU0102E: Promotion failed.

Explanation: The promote function failed to complete.

User Response: Check the configuration properties file has correct settings.

CWUDU0106E: Unable to commit transaction.

Explanation: The insertion of minimal entity data during the import function failed to commit to the database.

User Response: Check the database configuration. If necessary, turn on trace logging and look for the SQLException that is recorded.

CWUDU0107E: Unable to set auto-commit off on the database connection.

Explanation: UDDI Utility Tools needs to control commits of data changes, however the attempt to turn off auto-commit failed.

User Response: Check the database configuration. If necessary, turn on trace logging and look for the SQLExecption that is recorded.

CWUDU0109E: The import function requires a UDDI entity definition file to be specified.

Explanation: A required argument value was not supplied.

User Response: Specify -definition <path to Entity Definition File> on the command line, or set the value of the UDDIEntityDefinitionFile property in the configuration file to the path to the Entity Definition File.

CWUDU0110E: A cyclic dependency exists in the referenced tModels. The reference from tModel with key [<tModel key>] to the tModel with key [<tModel key>] completes the detected cycle.

Explanation: A cycle has been detected such that a tModel is being referenced by a tModel that it directly or indirectly references. This would cause the UDDI Utility Tools to enter an infinite loop trying to import referenced tModels, so the process is halted.

User Response: Edit the Entity Definition File and temporarily remove the reference to the tModel in the cycle, taking a note of the referenced details. After the import has successfully completed, update the tModel in the target registry to reintroduce the reference you previously removed. This can be done using the UDDI User Console, UDDI4J, or by creating a new Entity Definition File with just the tModel to be updated, and running the UDDI Utility Tools with the import function.

CWUDU0112E: An unexpected exception has occurred: <Exception message>.

Explanation: An unexpected error occurred.

User Response: Check configuration file settings and all registries and databases are active. If necessary, contact your IBM support center.

CWUDU0113E: Could not get a response from UDDI registry at URL: <URL>.

Explanation: A TransPortException occurred while performing an UDDI4J operation on the UDDI registry at the specified URL.

User Response: Check configuration properties for the UDDI registry in question and ensure the UDDI registry is active.

CWUDU0114E: An IOException occurred trying to invoke 'java'.

Explanation: When UDDI Utility Tools was invoked using the java -jar syntax, the invocation of the second JVM failed.

User Response: Check configuration property 'classpath' value is correct, and that Java is configured to run from the command line.

CWUDU0115I: Imported <entity count> entities and <referenced entity count> referenced entities.

Explanation: Indicate that the import step of the import or promote function has completed, showing the number of entities imported.

User Response: None.

CWUDU0116W: Not all minimal entities could be removed. The following remain in the database:

Explanation: A publish step was not successful which may have left one or more minimal entities in the target registry database. UDDI Utility Tools attempts to remove these minimal entities but in this case, the removal has failed. Following messages will indicate which minimal entities are left in the target registry.

User Response: You can attempt to remove the minimal entities using normal methods, such as the user console, UDDI4J, or using the delete function of the UDDI Utility Tools.

CWUDU0117W: Business minimal entities with businessKey [<business key>] has not been removed from the database.

Explanation: A business minimal entity was orphaned in the target registry database and attempts to remove it failed.

User Response: Identify the orphaned minimal entity in the target and attempt to remove using normal UDDI delete methods, or by using the delete function of the UDDI Utility Tools.

CWUDU0118W: Service minimal entity with serviceKey [<service key>] has not been removed from the database.

Explanation: A service minimal entity was orphaned in the target registry database and attempts to remove it failed.

User Response: Identify the orphaned minimal entity in the target registry and attempt to remove using normal UDDI delete methods, or by using the delete function of the UDDI Utility Tools.

CWUDU0119W: Binding Template minimal entity with bindingKey [<binding key>] has not been removed from the database.

Explanation: A binding minimal entity was orphaned in the target registry database and attempts to remove it failed.

User Response: Identify the orphaned minimal entity in the target registry and attempt to remove using normal UDDI delete methods, or by using the delete function of the UDDI Utility Tools.

CWUDU0120W: TModel minimal entity with tModelKey [<tModel key>] has not been removed from the database.

Explanation: A tModel minimal entity was orphaned in the target registry database and attempts to remove it failed.

User Response: Identify the orphaned minimal entity in the target registry and attempt to remove using normal UDDI delete methods, or by using the delete function of the UDDI Utility Tools.

CWUDU0121I: Created business minimal entity with businessKey [<business key>].

Explanation: Indicates the minimal data required for a businessEntity has successfully been inserted in the target UDDI registry database.

User Response: None.

CWUDU0122I: Created service minimal entity with serviceKey [<service key>].

Explanation: Indicates the minimal data required for a businessService has successfully been inserted in the target UDDI registry database.

User Response: None.

CWUDU0123I: Created binding template minimal entity with bindingKey [<binding key>].

Explanation: Indicates the minimal data required for a bindingTemplate has successfully been inserted in the target UDDI registry database.

User Response: None.

CWUDU0124I: Created tModel minimal entity with tModelKey [<tModel key>].

Explanation: Indicates the minimal data required for a tModel has successfully been inserted in the target UDDI registry database.

User Response: None.

CWUDU0125I: Deleted business minimal entity with businessKey [<business key>].

Explanation: Indicates the minimal data inserted for a businessEntity was successfully removed from the target UDDI registry database. This would normally happen after a publish operation has failed.

User Response: None.

CWUDU0126I: Deleted service minimal entity with serviceKey [<service key>].

Explanation: Indicates the minimal data required for a businessService was successfully removed from the target UDDI registry database. This would normally happen after a publish operation has failed.

User Response: None.

CWUDU0127I: Deleted binding template minimal entity with bindingKey [<binding key>].

Explanation: Indicates the minimal data required for a bindingTemplate was successfully removed from the target UDDI registry database. This would normally happen after a publish operation has failed.

User Response: None.

CWUDU0128I: Deleted tModel minimal entity with tModelKey [<tModel key>].

Explanation: Indicates the minimal data required for a tModel was successfully removed from the target UDDI registry database. This would normally happen after a publish operation has failed.

User Response: None.

CWUDU0129E: Find related businesses failed.

Explanation: The UDDI4J find related businesses operation did not complete.

User Response: Check the configuration properties for the source registry, such as fromInquiryURL.

CWUDU0130E: Find businesses failed.

Explanation: The UDDI4J find businesses operation did not complete.

User Response: Check the configuration properties for the source registry, such as fromInquiryURL.

CWUDU0131E: Find services failed.

Explanation: The UDDI4J find services operation did not complete.

User Response: Check the configuration properties for the source registry, such as fromInquiryURL.

CWUDU0132E: Find tModels failed.

Explanation: The UDDI4J find tModels operation did not complete.

User Response: Check the configuration properties for the source registry, such as fromInquiryURL.

CWUDU0133E: Find bindings failed.

Explanation: The UDDI4J find bindings operation did not complete.

User Response: Check the configuration properties for the source registry, such as fromInquiryURL.

CWUDU0134I: Performing inquiry request ...

Explanation: Indicated the find operation for selecting keys has started.

User Response: None.

CWUDU0135I: Extracted keys from inquiry results.
Explanation: Indicates the find operation to select keys has completed successfully.

User Response: None
CWUDU0136E: node ID value could not be found in UDDI database.
Explanation: The UDDI registries Node ID could not be found in the UDDI database.

User Response: Check that the UDDI application and database have initialized correctly.
CWUDU0137E: Unexpected database SQL exception: <SQL Exception Message>.
Explanation: An unexpected SQL Exception has been encountered.

User Response: Examine the SQL Exception Message to determine the cause of the problem.
CWUDU0138E: Could not delete minimal entity for tModel with tModelKey[<tModel Key>].
Explanation: The entity with the supplied tModel key could not be located and therefore deleted.

User Response: Ensure tModel key is correct.
CWUDU0139E: Could not delete minimal entity for Service with serviceKey[<Service Key>].
Explanation: The entity with the supplied Service key could not be located and therefore deleted.

User Response: Ensure Service key is correct.
CWUDU0140E: Could not delete minimal entity for Business with businessKey[<Business Key>].
Explanation: The entity with the supplied Business Key could not be located and therefore deleted.

User Response: Ensure Business key is correct.
CWUDU0141E: Could not delete minimal entity for Binding with bindingKey[<Binding Key>].
Explanation: The entity with the supplied Binding key could not be located and therefore deleted.

User Response:
CWUDU0142E: Invalid sequence number for service: <Sequence Number>.
Explanation: The ServiceStub has an invalid sequence number.

User Response: Ensure the sequence number is greater than zero.
CWUDU0143E: Invalid sequence number for binding: <Sequence Number>.
Explanation: The BindingStub has an invalid sequence number.

User Response: Ensure the sequence number is greater than zero.

CWUDVnnnns (Web Services UDDI Value Set Tools) messages:
CWUDV0001E: Unable to find the properties file: >Property File>.
Explanation: The named property file could not be located.

User Response: Supply the correct location of the property file.
CWUDV0002E: The column and string delimiters must not be the same.
Explanation: The column and string delimiters cannot be the same.

User Response: Supply different characters for the column and string delimiters.
CWUDV0003E: A tModel for key <tModel Key> can not be found.
Explanation: The tModel for the given key cannot be found.

User Response: Supply a correct tModel key.
CWUDV0004E: Invalid command arguments.
Explanation: A command argument has been supplied that is unknown.

User Response: Check your arguments with the Usage information given with this error message.
CWUDV0005E: Different tModel keys are required when using -newKey.
Explanation: Two unique tModel keys are required to move a Value Set from one tModel to another.

User Response: Supply two unique tModel keys.
CWUDV0006E: Unable to find the value set data file: <Value Set File>.
Explanation: The named value set file could not be located.

User Response: Supply the correct location of the value set file.

CWUDV0007E: There is an unterminated string on line <Line Number>:<Line>.

Explanation: The line at <Line Number> has a starting string delimiter, but not a finishing one.

User Response: Correct the line, or consider using a different string delimiter (-stringDelimiter)

CWUDV0008E: There were fewer fields than expected on line <Line Number>:<Line>.

Explanation: The line at <Line Number> does not contain enough fields.

User Response: Correct the line, of consider using a different column delimiter (-columnDelimiter).

CWUDV0009E: There were more fields than expected on line <Line Number>:<Line>.

Explanation: The line at <Line Number> contains too many fields.

User Response: Correct the line, or consider using a different column delimiter (-columnDelimiter).

CWUDV0010E: There is a duplicate Key Code at the same level on line <Line Number>.

Explanation: The Value Set file contains two or more Key Codes of the same value for the same parent key code.

User Response: Correct the line.

CWUDV0011E: An invalid Parent Key Code has been detected on line <Line Number>.

Explanation: The Value Set file contains a parent key code that is invalid in its context.

User Response: Correct the parent key code.

CWUDV0012E: The value set file contains a value <Column Contents> in column <Column Number> at line <Line Number> that is too long for the database table.

Explanation: The Value Set file contains a value that is too large.

User Response: Decrease the size of the value.

CWUDV0013E: An IO Exception has occurred: <IO Exception Message>.

Explanation: An IO Exception was received when trying to read the Value Set file.

User Response: Ensure the Value Set file is readable and is in UTF-8 format.

CWUDV0014E: There was a problem reading from the properties file: <IO Exception Message>.

Explanation: An IO Exception was received when trying to read the Value Set file.

User Response: Ensure the Value Set file is readable and is in UTF-8 format.

CWUDV0015E: The value set data file <Value Set File> is in an unsupported encoding.

Explanation: The Value Set file is not in UTF-8 format.

User Response: Correct the file encoding format.

CWUDV0016E: Unable to find the UDDI JMX MBean. Verify UDDI is installed.

Explanation: The UDDI application could not be contacted.

User Response: Ensure UDDI is installed and running on the Host you have targeted. See arguments -host, -port, -node and -server.

CWUDV0017E: An unexpected Exception has occurred.

Explanation: An unexpected exception was received.

User Response: Examine the Exception stack trace on its cause. If the problem cannot be resolver, contact your IBM Customer Service Center.

CWUDV1001W: The tModel with key <tModel Key> is checked. To confirm this operation, enter the command with the -override argument.

Explanation: The targeted tModel has a ″checked″ status and therefore should not have its value set changed without care.

User Response: To confirm this operation, enter the command with the -override argument.

CWUDV1002W: The tModel with key <tModel Key> has existing value sets. To confirm this operation, enter the command with the -override argument.

Explanation: The targeted to be loaded already has a value set.

User Response: To confirm this operation and overwrite the existing value set, enter the command with the -override argument.

CWUDV1003W: UDDI Registry Message: <Number of Lines> lines of data file for tModel key <tModel Key>.

Explanation: UDDI Registry message was received.

User Response: Examine the message on its cause. If the problem cannot be resolved, contact your IBM Customer Service Center.

CWUDV2001I: Loaded <Number of Lines> line of data file for tModel key <tModelKey>.

Explanation: An informational message that confirms the number of value set lines loaded for the tModel.

User Response:

CWUDV2002I: Changed value set from tModel key <tModel Key> to tModel key <tModel Key>.

Explanation: An informational message that confirms the change of value set from one tModel to another.

User Response: None.

CWUDV2003I: Unloaded value set for tModel key <tModel Key>.

Explanation: An informational message that confirms the removal of the value set for the tModel.

User Response: None.

CWUDXnnnns (Web Services JAXR) messages:

CWUDX0001E: Caught UDDIException on <UDDI API name>

Explanation: This is an Exception message. This message may be received in a RegistryException thrown by any of the following methods:

- Any method which necessitates sending a request to the UDDI registry.

The JAXR provider caught a org.uddi4j.UDDIException while sending a request to the UDDI registry.

User Response: The user should interrogate the cause UDDIException of the JAXRException for more information.

CWUDX0002E: Caught TransportException sending request to registry

Explanation: This is an Exception message. This message may be received in a RegistryException thrown by any of the following methods:

- Any method which necessitates sending a request to the UDDI registry.

The JAXR provider caught a org.uddi4j.TransportException while sending a request to the UDDI registry.

User Response: The user should interrogate the cause TransportException of the JAXRException for more information.

CWUDX0003E: AccessURI and TargetBinding are mutually exclusive

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- ServiceBinding.setAccessURI(String uri)
- ServiceBinding.setTargetBinding(ServiceBinding binding)

An attempt was made to set both the AccessURI and the TargetBinding of a ServiceBinding.

User Response: The user should set only one of AccessURI and TargetBinding.

CWUDX0004E: Source object of an Association must be an Organization

Explanation: This is an Exception message. This message may be received in an UnexpectedObjectException thrown by the following method:

- Association.setSourceObject(RegistryObject srcObject)

The object passed to the setSourceObject method was not an Organization.

User Response: The user should only pass Organization objects to the setSourceObject method.

CWUDX0005E: Source and target objects of an Association must be set in order to save

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- BusinessLifeCycleManager.confirmAssociation(Association assoc)
- BusinesslifeCycleManager.saveAssociations(Collection associations, boolean replace)
- LifeCycleManager.saveObjects(Collection objects) when objects are Associations

An attempt was made to save an Association that did not have both source and target objects set.

User Response: The user should only attempt to save Associations which have both source and target objects set.

CWUDX0006E: Target object of an Association must be an Organization

Explanation: This is an Exception message. This message may be received in an UnexpectedObjectException thrown by the following method:

- Association.setTargetObject(RegistryObject targetObject)

The object passed to the setTargetObject method was not an Organization.

User Response: The user should only pass Organization objects to the setTargetObject method.

CWUDX0007E: Format of associationKey is incorrect. Correct format is <sourceObjectKey>:<targetObjectKey>:<associationType> : <associationKey>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- BusinessLifeCycleManager.deleteAssociations(Collection associationKeys)

The user passed an associationKey to the deleteAssociations method that did not have the correct format.

User Response: The user should ensure that associationKeys passed to the deleteAssociations method have the correct format.

CWUDX0008E: AssociationType Concept must come from the AssociationType enumeration, and have value either HasChild , HasParent, RelatedTo or EquivalentTo

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- BusinessQueryManager.findAssociations(Collection findQualifiers, String sourceObjectId, String targetObjectId, Collection associationTypes)
- BusinessQueryManager.findCallerAssociations(Collection findQualifiers, Boolean confirmedByCaller, Boolean confirmedByOtherParty, Collection associationTypes)
- BusinessLifeCycleManager.confirmAssociation(Association assoc)
- BusinessLifeCycleManager.saveAssociations(Collection associations, boolean replace)
- LifeCycleManager.saveObjects(Collection objects) when objects are Associations

When finding Associations, the user passed a Concept in the associationTypes Collection that was from the AssociationType enumeration, but did not have a value that is valid for UDDI. When saving Associations, the user passed an Association whose associationType Concept was from the AssociationType enumeration, but did not have a value that is valid for UDDI.

User Response: The user should only use Concepts for associationTypes that are from the AssociationType enumeration and have value either HasChild, HasParent, RelatedTo or EquivalentTo.

CWUDX0009E: AssocationType Concept must come from the AssociationType enumeration

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- BusinessQueryManager.findAssociations(Collection findQualifiers, String sourceObjectId, String targetObjectId, Collection associationTypes)
- BusinessQueryManager.findCallerAssociations(Collection findQualifiers, Boolean confirmedByCaller, Boolean confirmedByOtherParty, Collection associationTypes)
- BusinessLifeCycleManager.confirmAssociation(Association assoc)
- BusinessLifeCycleManager.saveAssociations(Collection associations, boolean replace)
- LifeCycleManager.saveObjects(Collection objects) when objects are Associations

When finding Associations, the user passed a Concept that was not from the AssociationType enumeration in the associationTypes Collection. When saving Associations, the user passed an Association whose associationType Concept was not from the AssociationType enumeration.

User Response: The user should only use Concepts from the AssociationType enumeration for associationTypes.

CWUDX0010E: Cannot create a ClassificationScheme from a taxonomy Concept

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- LifeCycleManager.createClassificationScheme(Concept concept)

The user passed a taxonomy Concept to the createClassificationScheme method.

User Response: This method is provided to allow for Concepts returned by the BusinessQueryManager.findConcepts call to be safely converted to ClassificationScheme. It is up to the programmer to make sure that the Concept is indeed semantically a ClassificationScheme.

CWUDX0011E: Connection is closed

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- All methods of BusinessQueryManager and BusinessLifeCycleManager.
- The saveObjects and deleteObjects methods of LifeCycleManager.
- The saveObjects and deleteObjects methods of LifeCycleManager.

The user called a method that required a connection to the registry after they had closed the Connection by calling the Connection.close() method.

User Response: The user should not call methods that require a connection to the registry after the Connection has been closed.

CWUDX0012E: ConnectionFactory properties are not set

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- ConnectionFactory.createConnection()

The createConnection() method was called before the properties had been set on the ConnectionFactory.

User Response: Ensure that the ConnectionFactory properties have been set before attempting to create a Connection.

CWUDX0013E: Could not create DocumentBuilder

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by the following method:

- RegistryService.makeRegistrySpecificRequest(String request)

The JAXR provider caught a javax.xml.parsers.ParserConfigurationException while attempting to initialize the XML parser.

User Response: The user should interrogate the cause ParserConfigurationException of the JAXRException for more information.

CWUDX0014E: Could not parse XML input stream

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by the following method:

- RegistryService.makeRegistrySpecificRequest(String request)

The JAXR provider caught a java.io.IOException while attempting to parse the XML request.

User Response: The user should interrogate the cause IOException of the JAXRException for more information.

CWUDX0015E: Could not serialize XML response

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by the following method:

- RegistryService.makeRegistrySpecificRequest(String request)

The JAXR provider caught a java.io.IOException while attempting to serialize the XML response.

User Response: The user should interrogate the cause IOException of the JAXRException for more information.

CWUDX0016E: Interface name of object to create not specified

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- LifeCycleManager.createObject(String interfaceName)

The user passed a null interfaceName to the createObject method.

User Response: The user should ensure they only pass a valid interfaceName to the createObject method.

CWUDX0017W: Enumeration data file <filename> contains an invalid line: <line>

Explanation: This warning message will go to System.err if the JAXR provider encounters an invalid line in an enumeration data file while the Connection is initialized. The JAXR provider will ignore the invalid line. Otherwise the JAXR provider will be unaffected.

User Response: The user should ensure that the enumeration data file is valid in order to use all members of the enumeration. The correct format of each line is <enumeration name><separator char><concept value>.

CWUDX0018E: Could not read enumeration data file: <filename>

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by the following method:

- ConnectionFactory.createConnection(

The JAXR provider caught a java.io.IOException while attempting to read an enumeration data file.

User Response: The user should interrogate the cause IOException of the JAXRException for more information.

CWUDX0019W: enumerationConfig.properties file contains an invalid property value: <property value>

Explanation: This warning message will go to System.err if the JAXR provider encounters an invalid property value in the enumerationConfig.properties file while the Connection is initializsed. The JAXR provider will ignore the invalid property, and hence ignore the corresponding enumeration. Otherwise the JAXR provider will be unaffected.

User Response: The user should ensure that the enumerationConfig.properties file is valid in order to use all enumerations. The correct format of each line is <enumeration ID>=<enumeration name>,<data filename>,<separator char>

CWUDX0020E: An IOException occurred while attempting to read the enumerationConfig.properties file

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by the following method:

- ConnectionFactory.createConnection()

The JAXR provider caught a java.io.IOException while attempting to read the enumerationConfig.properties file.

User Response: The user should interrogate the cause IOException of the JAXRException for more information.

CWUDX0021E: An IOException occurred while attempting to read the taxonomyConfig.properties file

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by the following method:

- ConnectionFactory.createConnection()

The JAXR provider caught a java.io.IOException while attempting to read the taxonomyConfig.properties file.

User Response: The user should interrogate the cause IOException of the JAXRException for more information.

CWUDX0022E: External URI is malformed: <External URI>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- ExternalLink.setExternalURI(String uri)

A malformed URI was passed to the setExternalURI method when URI validation has set to true by passing true to the ExternalLink.setValidateURI(boolean validate) method.

User Response: Either the user should ensure that the URI is well formed, or URI validation should be set to false.

CWUDX0023E: External URI is not accessible: <External URI>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- ExternalLink.setExternalURI(String uri)

An inaccessible URI was passed to the setExternalURI method when URI validation has set to true by passing true to the ExternalLink.setValidateURI(boolean validate) method.

User Response: Either the user should ensure that the URI is accessible, or URI validation should be set to false.

CWUDX0024E: Invalid interface name: <interface name>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- LifeCycleManager.createObject(String interfaceName)

The user passed an invalid interface name to the createObject method.

User Response: The user should only pass valid interface names to the createObject method. Valid interface names are public final static String fields of the LifeCycleManager class.

CWUDX0025E: Cannot change the ClassificationScheme of an internal Classification

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- Classification.setClassificationScheme(ClassificationScheme classificationScheme)

The user called the setClassificationScheme method of an internal Classification.

User Response: The user should not attempt to modify the ClassificationScheme of an internal Classification directly. The ClassificationScheme of an internal Classification is determined by the Classification's Concept and cannot be modified independently.

CWUDX0026E: Cannot change the name of an internal Classification

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- Classification.setName(InternationalString name

The user called the setName method of an internal Classification.

User Response: The user should not attempt to modify the name of an internal Classification directly. The name of an internal Classification is determined by the Classification's Concept and cannot be modified independently.

CWUDX0027E: Cannot change the value of an internal Classification

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- Classification.setValue(String value)

The user called the setValue method of an internal Classification.

User Response: The user should not attempt to modify the value of an internal Classification directly. The value of an internal Classification is determined by the Classification's Concept and cannot be modified independently.

CWUDX0028E: The Concept of an internal Classification must have a parent ClassificationScheme

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- Classification.setConcept(Concept concept)

The user passed a non-null Concept without a parent ClassificationScheme to the setConcept method.

User Response: Setting a Classification's Concept causes a Classification to become internal. The Classification's ClassificationScheme is then set to the parent ClassificationScheme of the Concept. If the Concept has no parent ClassificationScheme (in other words, it is not a taxonomy Concept), this error will be encountered. The user should therefore only pass taxonomy Concepts to the setConcept method.

CWUDX0029E: Taxonomy Concepts cannot be saved as UDDI tModels

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:
- BusinessLifeCycleManager.saveConcepts(Collection concepts)
- LifeCycleManager.saveObjects(Collection objects) when the objects are Concepts.

The user attempted to save a taxonomy Concept as a UDDI tModel in the registry.

User Response: Taxonomy Concepts cannot be saved as tModels in a UDDI registry. They are used to classify objects saved in the registry but cannot be saved independently. The user should not attempt to save taxonomy Concepts in the registry.

CWUDX0030E: The parent RegistryObject of a taxonomy Concept must be either a Concept or a ClassificationScheme

Explanation: This is an Exception message. This message may be received in an UnexpectedObjectException thrown by any of the following methods:
- LifeCycleManager.createConcept(RegistryObject parent, InternationalString name, String value)
- LifeCycleManager.createConcept(RegistryObject parent, String name, String value)

The user attempted to create a taxonomy Concept whose parent was not a Concept or a ClassificationScheme.

User Response: The parent of a taxonomy Concept can only be another Concept or a ClassificationScheme, so the user should only attempt to set one of these as the parent of a taxonomy Concept.

CWUDX0031E: Concept does not have a parent, therefore does not have a path

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:
- Concept.getPath()

The user called the getPath() method on a Concept that was not a taxonomy Concept. Only taxonomy Concepts have parents.

User Response: Only taxonomy Concepts have parents, therefore only taxonomy Concepts have paths. The user should not attempt to call the getPath() method on a Concept that is not a taxonomy Concept.

CWUDX0032E: Concept does not have a value, therefore does not have a path

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:
- Concept.getPath()

The user called the getPath() method on a Concept that did not have a value.

User Response: A Concept must have a value in order to have a path, so the user should not attempt to call the getPath() method on Concepts that do not have a value.

CWUDX0033E: Concept's parent ClassificationScheme does not have an ID, therefore the Concept does not have a path

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:
- Concept.getPath()

The user called the getPath() method on a Concept whose ClassificationScheme did not have an ID.

User Response: A Concept's ClassificationScheme must have an ID in order for the Concept to have a path. The user should not attempt to call the getPath() method on a Concept whose ClassificationScheme does not have an ID.

CWUDX0034E: The ConnectionFactory property javax.xml.registry.uddi.maxRows does not contain a parsable integer:<javax.xml.registry.uddi.maxRows property value>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- ConnectionFactory.createConnection()

The user called the createConnection method when the ConnectionFactory property javax.xml.registry.uddi.maxRows did not contain a parsable integer.

User Response: The user should ensure that if the javax.xml.registry.uddi.maxRows ConnectionFactory property is set that it contains a parsable integer.

CWUDX0035E: Invalid UDDI XML String

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by the following method:

- RegistryService.makeRegistrySpecificRequest(String xmlString)

The String passed to the makeRegistrySpecificRequest method was not valid XML.

User Response: The user should ensure that the String passed to the makeRegistrySpecificRequest method is valid XML.

CWUDX0036E: The ConnectionFactory property javax.xml.registry.lifeCycleManagerURL specifies a malformed URL

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- ConnectionFactory.createConnection()

The user called the createConnection method when the ConnectionFactory property javax.xml.registry.lifeCycleManagerURL contained a malformed URL.

User Response: The user should ensure that the javax.xml.registry.lifeCycleManagerURL ConnectionFactory property contains a well formed URL.

CWUDX0037E: The ConnectionFactory property javax.xml.registry.queryManagerURL specifies a malformed URL

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- ConnectionFactory.createConnection()

The user called the createConnection method when the ConnectionFactory property javax.xml.registry.queryManagerURL contained a malformed URL.

User Response: The user should ensure that the javax.xml.registry.queryManagerURL ConnectionFactory property contains a well formed URL.

CWUDX0038E: Multiple matches on find ClassificationScheme by name

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- BusinessQueryManager.findClassificationSchemeByName(Collection findQualifiers, String namePattern)

More than one ClassificationScheme was found that matched the search criteria.

User Response: The user should narrow their search criteria to find only one ClassificationScheme.

CWUDX0039E: Invalid objectType: <object type>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- LifeCycleManager.deleteObjects(Collection keys, String objectType)
- QueryManager.getRegistryObjects(String objectType)
- QueryManager.getRegistryObject(String id, String objectType)
- QueryManager.getRegistryObjects(Collection objectKeys, String objectType)

The user passed an invalid objectType to one of the above methods.

User Response: The user should ensure they pass a valid objectType to the above methods. The valid objectTypes for these methods are:

```
LifeCycleManager.CLASSIFICATION_SCHEME
LifeCycleManager.CONCEPT
LifeCycleManager.ORGANIZATION
LifeCycleManager.SERVICE
LifeCycleManager.SERVICE_BINDING
```

The deletObjects method also accepts an objectType of LifeCycleManager.ASSOCIATION.

CWUDX0040E: Cannot save objects of type: <object class>

Explanation: This is an Exception message. This message may be received in an UnexpectedObjectException thrown by the following method:

- LifeCyclemanager.saveObjects(Collection objects)

An object was passed to the saveObjects method of a type that cannot be saved directly in the registry.

User Response: The user should ensure that objects passed to the saveObjects method are of a valid type. Valid types are Association, ClassificationScheme, Concept, Organization, Service and ServiceBinding.

CWUDX0041E: RegistryObject is a ClassificationScheme not a Concept: <RegistryObject ID>

Explanation: This is an Exception message. This message may be received in a FindException thrown by any of the following methods:

- QueryManager.getRegistryObject(String id, String objectType)
- QueryManager.getRegistryObjects(Collection objectKeys, String objectType)

An objectType of LifeCycleManager.CONCEPT was passed to one of the above methods, but the id or one of the objectKeys was that of a ClassificationScheme.

User Response: The user should ensure that they specify the correct objectType corresponding to the object keys.

CWUDX0042E: RegistryObject is a Concept not a ClassificationScheme: <RegistryObject ID>

Explanation: This is an Exception message. This message may be received in a FindException thrown by any of the following methods:

- QueryManager.getRegistryObject(String id, String objectType)
- QueryManager.getRegistryObjects(Collection objectKeys, String objectType)

An objectType of LifeCycleManager.CLASSIFICATIONSCHEME was passed to one of the above methods, but the id or one of the objectKeys was that of a Concept.

User Response: The user should ensure that they specify the correct objectType corresponding to the object keys.

CWUDX0043E: RequestID not found: <RequestID>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- RegistryService.getBulkResponse(String requestId

The requestId specified was not found.

User Response: The user should only pass valid requestIds to the getBulkResponse method. Note that once the getBulkResponse method has been called once for a particular requestId, that requestId is removed from the cache and subsequent calls to getBulkResponse passing that requestId will result in an InvalidRequestException.

CWUDX0044E: <concept path> is not a valid path of a concept in a defined internal taxonomy

Explanation: This message will go to System.err when a Connection is created if the javax.xml.registry.semanticEquivalences ConnectionFactory property defines a semantic equivalence between a Concept in the PostalAddressAttributes enumeration and a Concept which has not been defined in any internal taxonomy.

User Response: The user should ensure that the Concept paths used in the javax.xml.registry.semanticEquivalences ConnectionFactory property have been defined in a internal taxonomy.

CWUDX0045W: Semantic equivalence pair does not have exactly 2 elements: <keyPair>

Explanation: This message will go to System.err when a Connection is created if the javax.xml.registry.semanticEquivalences ConnectionFactory contains a keyPair which contains more than two elements.

User Response: The user should ensure that the javax.xml.registry.semanticEquivalences ConnectionFactory property has the correct format, as defined in the JAXR specification.

CWUDX0046E: Semantic equivalence pair does not contain a key in the postalAddressAttributes enumeration: <keyPair>

Explanation: This message will go to System.err when a Connection is created if the javax.xml.registry.semanticEquivalences ConnectionFactory contains a keyPair which does not contain the path of a Concept in the PostalAddressAttributes enumeration. Semantic equivalences for a UDDI JAXR providers are only allowed for Concepts in the PostalAddressAttributes enumeration.

User Response: The user should only attempt to define semantic equivalences for Concepts in the PostalAddressAttributes enumeration.

CWUDX0047E: Invalid Slot name: <Slot name>

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- All methods of the ExtensibleObject interface.

The user passed an invalid slot name to one of the methods of the ExtensibleObject interface.

User Response: The user should ensure that the slot name is valid for the particular instance of ExtensibleObject.

CWUDX0048E: A Slot instance cannot have duplicate values

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- Slot.setValues(Collection values)

The user passed a collection of values to the setValues method that contained duplicate values.

User Response: The user should pass only a collection of unique values to setValues method.

CWUDX0049E: A sortCode Slot must have only 1 value

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- PostalAddress.addSlot(Slot slot)

The user passed a Slot with name Slot.SORT_CODE_SLOT and multiple values to the addSlot method.

User Response: When adding a Slot with name Slot.SORT_CODE_SLOT to a PostalAddress, the user should ensure that it only has 1 value.

CWUDX0050E: A specificationLink can only have one ExternalLink

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- SpecificationLink.addExternalLink(ExternalLink externalLink)
- SpecificationLink.addExternalLinks(Collection externalLinks)
- SpecificationLink.setExternalLinks(Collection externalLinks)

The user attempted to give a SpecificationLink more than one ExternalLink. A SpecificationLink may only have one ExternalLink.

User Response: The user should give a SpecificationLink a maximum of one ExternalLink.

CWUDX0051E: A SpecificationLink can only have one UsageParameter

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:

- SpecificationLink.setUsageParameters(Collection usageParameters)

The user attempted to give the SpecificationLink more than one usage parameter. A SpecificationLink can only have one usage parameter.

User Response: The user should give a SpecificationLink a maximum of one usage parameter.

CWUDX0052E: SpecificationObject must be a Concept with no parent

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by any of the following methods:

- SpecificationLink.setSpecificationObject(RegistryObject obj)

The user attempted to set a Concept with a parent (that is, a taxonomy Concept) as the specification object of the SpecificationLink.

User Response: The user must set a specification Concept as the specification object of a SpecificationLink.

CWUDX0053E: SpecificationObject must be a Concept

Explanation: This is an Exception message. This message may be received in an UnexpectedObjectException thrown by the following method:

- SpecificationLink.setSpecificationObject(RegistryObject obj)

The user attempted to set a RegistryObject that was not a Concept as the specification object of a SpecificationLink.

User Response: The user must set a specification Concept as the specification object of a SpecificationLink.

CWUDX0054E: Invalid escape sequence found during SQL-92 LIKE Processing: <escape sequence>

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by any of the following methods:

- BusinessQueryManager.findClassificationSchemeByName(Collection findQualifiers, String namePattern)
- BusinessQueryManager.findClassificationSchemes(Collection findQualifiers, Collection namePatterns, Collection classifications, Collection externalLinks)

The user passed a namePattern to one of the above methods which contained an invalid escape sequence.

User Response: The user should ensure that namePatterns do not contain invalid escape sequences.

CWUDX0055E: Invalid escape sequence found terminating pattern during SQL-92 LIKE processing: <escape sequence>

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by any of the following methods:

- BusinessQueryManager.findClassificationSchemeByName(Collection findQualifiers, String namePattern)
- BusinessQueryManager.findClassificationSchemes(Collection findQualifiers, Collection namePatterns, Collection classifications, Collection externalLinks)

The user passed a namePattern to one of the above methods which contained an invalid escape sequence terminating the pattern.

User Response: The user should ensure that namePatterns do not contain invalid escape sequences.

CWUDX0056E: The System property http.proxyPort does not contain a parsable integer: <Value of http.proxyPort property>

Explanation: This is an Exception message. This message may be received in a java.lang.NumberFormatException thrown by the following method:

- ConnectionFactory.createConnection()

The user called the createConnection() method when the System property http.proxyPort contained a String that was not parsable as an integer.

User Response: The user should ensure that if the System property http.proxyPort is set it contains a parsable integer.

CWUDX0057W: Taxonomy data file <filename> contains an invalid line: <invalid line>

Explanation: This message will go to System.err when a Connection is created if a taxonomy data file contains an invalid line.

User Response: The format of each line is <taxonomy ID><Concept name><Concept value><Concept parent>

CWUDX0058W: Warning: Unable to locate parentConcept named <parent Concept name> for concept named <Concept name> in taxonomy datafile <filename>

Explanation: This message will go to System.err when a Connection is created if a taxonomy data file contains an line for a Concept whose parent cannot be located in that file.

User Response: The user should ensure that a parent exists for each Concept in the taxonomy data file.

CWUDX0059E: Could not read taxonomy data file: <filename>

Explanation: This is an Exception message. This message may be received in a JAXRException thrown by any of the following methods:

- ConnectionFactory.createConnection()

The JAXR provider caught a java.io.IOException while attempting to read the taxonomy data file.

User Response: The user should interrogate the cause IOException of the JAXRException for more information.

CWUDX0060W: taxonomyConfig.properties file contains an invalid property value: <property value>

Explanation: This warning message will go to System.err if the JAXR provider encounters an invalid property value in the taxonomyConfig.properties file while the Connection is initialized. The JAXR provider will ignore the invalid property, and hence ignore the corresponding taxonomy. Otherwise the JAXR provider will be unaffected.

User Response: The user should ensure that the taxonomyConfig.properties file is valid in order to use all taxonomies. The correct format of each line is <taxonomy ID>=<tModelKey>,<data filename>,<separator char>.

CWUDX0061E: Expecting object of type: <objectType String>. Got object of type: <object class>

Explanation: This is an Exception message. This message may be received in an UnexpectedObjectException thrown by the following method:

- All methods which accept objects of ambiguous type.

The user passed an object to a method that was not expecting an object of that type.

User Response: The user should only pass objects of the appropriate type to JAXR methods.

CWUDX0062E: Expecting object of type String or LocalizedString. Got object of type: <object class>

Explanation: This is an Exception message. This message may be received in an UnexpectedObjectException thrown by any of the following methods:

- All query methods which accept a Collection of namePattern objects.

The user passed an object which was not a String or a LocalizedString as a namePattern to a query method.

User Response: The user should only use Strings of LocalizedStrings as namePattern objects.

CWUDX0063E: The ConnectionFactory property javax.xml.registry.queryManagerURL is not specified

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:
- ConnectionFactory.createConnection(

The user attempted to create a Connection without specifying the javax.xml.registry.queryManagerURL ConnectionFactory property.

User Response: The user must specify the javax.xml.registry.queryManagerURL ConnectionFactory property before attempting to create a Connection.

CWUDX0064E: Unsupported value for the ConnectionFactory property

Explanation: This is an Exception message. This message may be received in an InvalidRequestException thrown by the following method:
- ConnectionFactory.createConnection()

The user attempted to create a Connection with an invalid value of the javax.xml.registry.security.authenticationMethod ConnectionFactory property.

User Response: The user should only use a valid vale for this property. Valid values are UDDI_GET_AUTHTOKEN and HTTP_BASIC.

UDDI Registry samples

The UDDI samples, and documentation on how to use them, are available through the Web Services UDDI samples link on the Samples Central page of the IBM WebSphere Developer Domain Web site.

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 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 IBM 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 IBM 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 IBM 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 482
2. Review the "UDDI Registry security additional considerations" on page 483.

Configuring the UDDI Registry to use WebSphere Application Server security:

Before starting this task complete the following two steps:

- Enable WebSphere Application Server global security (see Configuring global security). 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, please refer to Secure Sockets Layer settings for custom properties.

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 uses Role Mappings to determine authorization. UDDI makes use of two special WebSphere Application Server roles: 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 role, and data confidentiality is enforced (HTTPS is used). The services in question are as follows:
  - Versions 1 and 2 SOAP publish service (SOAP_Publish_User)
  - EJB publish service (EJB_Publish_Role)
  - Version 3 GUI publish service (GUI_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)

  Authentication uses the standard WebSphere security facilities and there is no separate registration function for the UDDI registry. You will need to supply your WebSphere user name and password for publish functions (unless you have modified the supplied publish role).
- Inquiry services are mapped to the Everyone role, and data confidentiality is not enforced (HTTP is used). The UDDI inquiry services are as follows:
  - Versions 1 and 2 SOAP inquiry service (SOAP_Inquiry_User)
  - EJB inquiry service (EJB_Inquiry_Role)

– Version 3 GUI inquiry service (GUI_Inquiry_User)
– Version 3 SOAP inquiry service (V3SOAP_Inquiry_User_Role)

No further configuration is necessary. However, if you wish to change the default settings follow the steps below:

1. To change the role mappings, use the administrative console to 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 **Additional Properties** on the right hand side, click **Map security roles to users/groups**.
   d. Make any changes you require and click **OK**.
2. To change the data confidentiality settings, refer to “Configuring SOAP API and GUI services” on page 485.

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 483

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 **Additional Properties** on the right hand side, click **Map security roles to users/groups**.
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 **APIs**. 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 481.

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*. In the **General Properties** section select the **Use authInfo credentials if provided** check box.
2. Ensure that UDDI Policy is set to require the use of authentication tokens for the UDDI Publish and Custody Transfer services: click **UDDI** → **UDDI Nodes** > *uddi_node_name*, and under **Policy Groups** click **APIs**. 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).

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

The UDDI Registry also supports 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″ in the information center.

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 **APIs**. 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. If WebSphere Application Server security is enabled and 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 and Configuring UDDI Security with WebSphere Application Server security disabled.

Other considerations

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 **UDDI Keying**. 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.

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:

   ```
   install_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:

   ```
   install_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. Modify the user data constraint transport guarantee which determines whether the service is to be confidential (accessed using HTTPS) or insecure (accessed using HTTP). Use the table below to find the value of 'security-constraint id' for the service you wish to modify. Find the relevant security-constraint id in the deployment descriptor and set its 'user-data-constraint' 'transport-guarantee' to either CONFIDENTIAL or NONE.

Type of UDDI service	Value of security-constraint id
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:

   ```
   install_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. Modify the user data constraint transport guarantee which determines whether the service is to be confidential (accessed using HTTPS) or insecure (accessed using HTTP). Use the table below to find the value of 'security-constraint id' for the service you wish to modify. Find the relevant security-constraint id in the deployment descriptor and set its 'user-data-constraint' 'transport-guarantee' to either CONFIDENTIAL or NONE.

Type of UDDI service	Value of security-constraint id
Publish	UDDIPublishSecurityConstraint
Inquiry	UDDIInquireSecurityConstraint

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:
install_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, all policies and properties are managed 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 IBM WebSphere Registry Administrative 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

UDDI node collection

Start the associated UDDI application for this node, if it is not already running. If the application is not running you will not see the UDDI node in the list of available choices.

To configure node properties, policies, value set status and user entitlements, complete the following steps:

From the administrative console, expand **UDDI** in the navigation pane then click **UDDI Nodes**. This displays the collection of UDDI nodes in the cell.

Each UDDI node is represented by a UDDI Node ID, Description, UDDI Application Location and Status. The Status can be either *Initialization Pending*, *Activated* or *Deactivated*. To activate UDDI nodes that are *Deactivated*, select them by checking the corresponding check boxes in the Select column and click the *Activate* button. Similarly to deactivate UDDI nodes, select them and click the *Deactivate* button.

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.

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. Refer to UDDI node settings for details on the next available topic.

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 UDDI node policy may be viewed and changed.

Unless the UDDI node has been installed as a default UDDI node (as defined in UDDI Registry terminology) there are some important general properties that need to be set before the UDDI node can be initialized. These properties are all 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, 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. Other general properties of the UDDI node may be set before, and after, initialization.

Once the general properties have been set to appropriate values, you can click *OK* (which saves your changes and exits the page), or *Apply* (which saves your changes and leaves you on the same page). At this point the changes will have been 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), the UDDI node can be initialized by clicking the *Initialize* button. This operation may take a while to complete. It is important to remember to save any changes you have made to the general properties by clicking *Apply* or *OK* before the *Initialize* button is pressed.

The other UDDI settings that a UDDI administrator can manage are shown to the right of the screen and are described in the following topics:

- Value set collection

  This topic contains details of the value sets settings that you can configure for a UDDI node.
- Tier collection

  This topic contains details of the UDDI publisher tiers that you can configure for a UDDI node.
- UDDI Publisher collection

  This topic contains details of the publishers that have been registered with the UDDI node.
- Policy groups

  This topic contains links to the detailed settings information for every policy group that you can configure for a UDDI node.

UDDI Node ID:

The unique identifier given to a UDDI node in a UDDI registry. This must be a valid UDDI key.

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:

The user supplied 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 may 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:

The URL prefix for the GET servlet.

The URL prefix applied to generated discoveryURLs in businessEntity elements so they can be returned on HTTP GET requests. The format is 'http://hostname:port/uddisoap/', where 'uddisoap' is the UDDI version 2 SOAP servlet's context root. This property applies to UDDI version 2 API requests only.

Required Yes
Data type String
Default http://localhost:9080/uddisoap

Host name for UDDI node services:

The fully qualified domain name of the network host, or its IP address.

The hostname root used by the UDDI node to model API services in its own node business entity.

Data type String
Default localhost

Host HTTP port:

The port number used to access UDDI node services with HTTP.

This port number must match the WebSphere port for HTTP requests.

Data type Integer
Default 9080

Host HTTPS port:

The port number used to access UDDI node services with HTTPS.

This port number must match the WebSphere port for HTTPS requests.

Data type Integer
Default 9443

Maximum inquiry result set size:

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. Setting the value higher allows larger result sets but may cause increased response times.

CAUTION: If this value is set too low users will get an E_resultSetTooLarge error message, whereas setting to too high might cause increased response times. If the value is set too low and users use imprecise search criteria the likelihood of receiving the E_resultSetTooLarge is increased.

Data type Integer
Default 500

Range	0 to 1024

Maximum inquiry response set size:

For inquiry API requests, this value controls the maximum number of results returned in each response. If the number of results in the result set is greater than this value, the response will only include a subset of results. The user can retrieve remaining results using the listDescription feature as described in the UDDI specification. Setting this value too low will require the user to make more requests to retrieve the remainder of the result set.

CAUTION: This value can not be higher than the value set for ″Maximum inquiry results″. Setting this value too low increases the number of requests needed to achieve the full result set.

Data type	Integer
Default	500
Range	0 to 1024

Maximum search names:

The maximum number of names that can be supplied in an inquiry API request. Increasing this value can significantly slow response times of the UDDI node.

This can be used to control the complexity of requests that this UDDI node will allow. The recommendation is to not set this value above 8.

Data type	Integer
Default	5
Range	1 to 64

Maximum search keys:

The maximum number of keys that can be supplied in an inquiry API request. This limits the number of references that can be specified in categoryBag, identifierBag, tModelBag and discoveryURLs. Increasing this value can significantly increase response times for the UDDI node.

This can be used to control the complexity of requests that this UDDI node will allow. The recommended setting for this is 5 or less.

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 set 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 if authInfo contents in UDDI API requests are used to validate users when WebSphere global security is off. If this setting is true, the UDDI node will use the request's authInfo element, otherwise the default user name is used.

Data type	Boolean (check box)
Default	True (selected)

Authentication token expiry period:

The period after which authentication tokens are invalidated (in minutes), and a new authToken is required.

CAUTION: The setting should be sufficient to ensure operational success. Longer settings can increase the risk of illegal authToken use.

Data type	Integer
Default	30
Range	1 to 10080 minutes (10080 minutes = 1 week)

Automatically register UDDI publishers:

Specifies if 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 security is disabled.

Data type	String
Default	UNAUTHENTICATED

Default language code:

Applies only to 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 have been installed in a UDDI node.

For information on adding new value sets please see User Defined Value Set Support in the UDDI Registry.

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. When you have published 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 clicking on the appropriate check boxes in the *Select* column. Click the *Enable Support* button. The supported field for all the selected value sets will be updated, with a value of true, to reflect the new status.

To disable support for a value set, which may be necessary before it is removed from the UDDI node, select the value sets in the same manner as for enabling support. Click the *Disable Support* button. The supported field for all the selected value sets will be 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:

The name of the tModel that represents the value set.

tModelkey:

The key for the tModel that represents the value set.

Supported:

Supported Indicates whether references to this value set are supported (true) or not supported (false) 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. It also shows the *Supported* status of the value set as described on the Value set collection page. All properties are read-only. The *Supported* status can be changed on the Value Set page.

Unvalidatable:

Specifies whether this value set is unvalidatable. This is set by the value set tModel publisher to indicate if the value set is available or not for use by publish requests.

Checked:

Specifies whether this value set is checked. If set to true, UDDI entities that reference this value set will be validated to ensure 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:

The name of the tier.

Description:

User supplied 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:

The name of the tier.

Required Yes

Data type	String
Default	No default
Range	1 to 255

Description:

The user supplied description of the 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:

The maximum number of businesses that UDDI publishers in this tier are allowed to publish in this tier.

Maximum services:

The maximum number of services UDDI publishers in this tier are allowed to publish.

Maximum bindings:

The maximum number of bindings UDDI publishers in this tier are allowed to publish.

Maximum tModels:

The maximum number of tModels UDDI publishers in this tier are allowed to publish.

Maximum publisher assertions:

The maximum number of publisher assertions UDDI publishers in this tier are allowed to add.

UDDI Publisher collection:

This page shows the WebSphere users that are currently registered as UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* >**UDDI Publishers**.

To create a UDDI publisher click on the New button. This opens the UDDI Publisher settings page where details about the publisher can be entered.

It is possible to assign multiple publishers to a tier without having to edit each one individually. To do this select the appropriate publishers in the collection table. From the selection box at the top of the collection table choose from one of the tiers available on the UDDI node. Finally, click the *Assign tier* button to update the selected publishers.

To delete publishers select them in the collection table and then press the *Delete* button.

After the users have been registered as UDDI publishers, their entitlements can be edited as described in UDDI Publisher settings.

User name:

The name of the UDDI publisher.

Tier:

The publication limits tier to which the UDDI publisher has been 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:

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:

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:

The UDDI publisher has permission to publish tModel:keyGenerator with a domain key.

Data type	Boolean
Default	True (selected)

Allowed to publish keyGenerator:

The UDDI publisher has permission to publish tModel:keyGenerator.

If false, UDDI publishers cannot publish keyGenerators of any kind. In this situation the following permissions ('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') will be disregarded irrespective of how they are set.

Data type	Boolean
Default	True (selected)

Allowed to publish with UUID key:

The UDDI publisher has permission to publish elements providing a UUID key.

Data type	Boolean
Default	False (cleared)

Allowed to publish keyGenerator with UUID keys:

The UDDI publisher user has permission to publish tModel:keyGenerator providing a UUID key.

Data type	Boolean
Default	False (cleared)

Tier:

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.

The policy groups available to act upon are:
- "UDDI keying policy settings"
- "UDDI node API policy settings" on page 497
- "UDDI user policy settings" on page 497
- "UDDI data custody policy settings" on page 498
- "UDDI value set policy" on page 498
- "UDDI node miscellaneous" on page 499

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* >**UDDI Keying**.

Registry key generation:

Allows publishers to publish key generator tModels.

Defines whether a given UDDI node or publisher is allowed to register a key generator tModel. When true, this allows the set of publishers to be managed using the facilities provided in the UDDI Publisher settings

Data type	Boolean
Default	True (selected)

Registry support of UUID keys:

Allow publisher supplied uuidKeys in publish requests.

If true, this allows the set of publishers to be managed using the facilities provided in UDDI Publisher settings

Data type	Boolean
Default	False (cleared)

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 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* > **APIs**.

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 if authorization using the authInfo element is required for inquiry API requests.

Typically, UDDI registries are configured not to require authorization for registry API requests. This setting is only relevant if the V3SOAP_Inquiry_User_Role is set to *everyone* and WebSphere Application Server global security is on. If WebSphere Application Server global security is off, this setting is ignored. If WebSphere Application Server global security is on and the V3SOAP_Inquiry_User_role is not set to *everyone*, this setting is ignored.

Data type	Boolean
Default	False (cleared)

Authorization for publish:

Specifies if authorization using the authInfo element is required for publish API requests.

Typically, UDDI registries are configured not to require authorization for registry API requests. This setting is only relevant if the V3SOAP_Publish_User_Role is set to *everyone* and WebSphere Application Server global security is on. If WebSphere Application Server global security is off, this setting is ignored. If WebSphere Application Server global security is on and the V3SOAP_Publish_User_role is not set to *everyone*, this setting is ignored.

Data type	Boolean
Default	True (selected)

Authorization for custody transfer:

Specifies if authorization using the authInfo element is required for custody transfer API requests.

Typically, UDDI registries are configured not to require authorization for registry API requests. This setting is only relevant if the V3SOAP_CustodyTransfer_User_Role is set to *everyone* and WebSphere Application Server global security is on. If WebSphere Application Server global security is off, this setting is ignored. If WebSphere Application Server global security is on and the V3SOAP_CustodyTransfer_User_role is not set to *everyone*, this setting is ignored.

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**.

Transfer token expiration period:

The length of time (in minutes) after the issue of a transfer token before it expires.

CAUTION: Setting too large a value 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 Policy**.

Enable checked value sets:

Specifies if checked value sets are supported. When false, publish requests containing references to checked value sets are 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**.

Node generates discoveryURLs:

Defines whether the UDDI node generates discoveryURLs.

Data type	Boolean
Default	False (cleared)

Node supports HTTP Get Service:

Specifies if 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:

The URL prefix for the UDDI version 3 GET servlet

The prefix for the URL to the GET servlet used to retrieve the XML representation of a published entity. When a businessEntity is published, if the policy for Node Discovery URLs is set to true, the discoveryURL value is generated based on the prefix value. Otherwise, the discoveryURL value will be empty.

The UDDI Version 3 specification recommends that discoveryURLs are **not** generated as they can affect the use of digital signatures. If you do enable generation of discoveryURLs, it is recommended that the URL prefix is not changed after the point at which the policy to enable generation of discoveryURLs is enabled. Not doing so will mean discoveryURLs generated using the earlier URL prefix would no longer work.

Data type	URL
Default	http://localhost:9080/uddiv3soap/

UDDI Registry Administrative Interface Overview

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 IBM WebSphere 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, 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, then make a copy of the UDDI30 directory using the file system that the directory resides upon.

To restore a Cloudscape 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 import and export tools for the database being used to contain the UDDI Registry.

- Backup

  Include elements in schemas named IBMUDI30 and IBMUDS30
- Restore

  Restore the back up by deleting the schemas IBMUDI30 and IBMUDS30, recreating database structures using the original scripts (with slight modifications), and importing the previously saved data, as described in the steps below:

  1. Delete the schemas IBMUDI30 and IBMUDS30 *this will result in any IBM UDDI structures being destroyed.*
  2. Create database structures (for DB2 see Creating a DB2 database or for Cloudscape, see Creating a Cloudscape database for details) as for the original creation **except** that the final step (xxxxxx_**70**xxxx.*) must not be run. This will result in a database containing the basic data required by the UDDI Registry, into which your backed-up data may be imported.
  3. Delete all the rows in the table IBMUDS30.UDDIDBSCHEMAVER, to avoid a clash between a row inserted by the scripts and the row that was backed-up. To do this, use the following command:

     ```
     delete from IBMUDS30.UDDIDBSCHEMAVER
     ```
  4. Restore the previously backed-up data in schemas IBMUDI30 and IBMUDS30

Removing and reinstalling the UDDI Registry

An IBM WebSphere UDDI Registry node consists of a WebSphere application, a store of data (using a relational database management system or RDBMS) 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.

Completely removing a UDDI Registry node from an application server
You might want to do this in order to move the UDDI Registry to a different application server, or 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 *install_root*/bin directory.

   The syntax of the command is as follows:

   ```
   wsadmin.sh -f uddiRemove.jacl
                node_name
                server_name
                [default]
   ```

   6.1+

   ```
   wsadmin.sh -f uddiRemove.jacl
                {node_name server_name | cluster_name}
                [default]
   ```

   where

   - *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).
   - 6.1+ *cluster_name* is the name of the WebSphere cluster in which the UDDI application is deployed. This is the name that you specified when you ran uddiDeploy.jacl to install the UDDI application.
   - 'default' is optional and is applicable only for Cloudscape in a standalone application server environment, 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' and send any messages to the file removeuddi.log:

   ```
   wsadmin.sh -f uddiRemove.jacl MyNode server1 > removeuddi.log
   ```

   6.1+ To remove the UDDI application from cluster 'MyCluster' and send any messages to the screen:

   ```
   wsadmin.sh -f uddiRemove.jacl MyCluster
   ```

Note: If running in a network deployment configuration, the command must be executed against the deployment manager profile.

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 Cloudscape, delete the directory tree containing the UDDI database. By default, this will be located in *install_root*/profiles/*profile_name*/databases/com.ibm.uddi/UDDI30.

3. **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. Run the uddiRemove.jacl script as described above, to remove the UDDI Registry application.

   b. Delete the UDDI Registry database, as described above.

   c. If you ran the uddiRemove 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 *install_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 are deploying the UDDI Registry into a network deployment scenario, ensure that the deployment manager is the target.

   At a command prompt enter:

```
wsadmin.sh [-conntype none] –f uddiDeploy.jacl
              node_name
              server_name
```

6.1 +

```
wsadmin.sh [-conntype none] –f uddiDeploy.jacl
              {node_name server_name | cluster_name}
```

   where

   - '-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).

- 6.1+ *cluster_name* is the name of the WebSphere cluster in which the UDDI application is deployed. This is the name 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.

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. See ″Developing 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. See ″Assembling data access applications″ in the information center. 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 (RRA) 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 version 6.0 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.

Resource Recovery Services (RRS):

WebSphere Application Server for z/OS supports resource adapters that use Resource Recovery Services (RRS) to support global transaction processing. RRS is an z/OS extension to the JCA resource adapter specifications.

WebSphere Application Server for z/OS supports the J2EE Connector Architecture (JCA) 1.0., and because of this, any resource adapter that is designed to use the 1.0 level of the J2EE Connector Architecture (JCA) is supported.

In addition to the 3 types of transaction support defined by JCA, WebSphere Application Server for z/OS supports a fourth type, **RRSTransactional** support, which is a z/OS only extension to the architecture. Resource adapters that are capable of using RRS and that properly indicate to WAS z/OS they are RRSTransactional will be supported as RRS compliant resource adapters.

z/OS resource adapters that are capable of using RRS are:
- IMS Connector for Java
- CICS CTG ECI J2EE Connector
- IMS JDBC Connector

- DB2 for z/OS Local JDBC connector when used as aJDBC Provider under the WebSphere Relational Resource Adapter (RRA)

All RRS Compliant resource adapters are required to support the property **RRSTransactional** in their ManagedConnectionFactory and must support a getter method for the property.

```
java.lang.Boolean.RRSTransactional=true;

java.lang.Boolean getRRSTransactional(){
    // Determine if the adapter can run RRSTransactional based
   // on it's configuration, and set the RRSTransactional property
   // appropriately to true or false.
 return RRSTransactional;
}
```

RRS support is only applicable in a local environment, where the backend must reside on the system. CICS and IMS resources adapters may use **RRSTransactional** support only when these adapters are configured to use local interfaces to their backend resource manager, which as stated above must reside on the same system as the IBM WebSphere Application Server for z/OS. These adapters are also capable of being configured to a remote instance of their backend resource manager. In this case, the adapters will respond ″false″ when the getRRSTransactional() method is invoked and instead of running as RRSTransactional will use whichever one of the three types of J2EE Transaction support they have chosen to support.

WebSphere relational resource adapter settings:

Use this page to view the default WebSphere relational resource adapter settings.

This is the WebSphere-provided relational resource adapter for handling data access to any relational data base. This adapter is preinstalled by WebSphere Application Server. 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** > **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 503
- For information about resource adapters and data access, see “Data access portability features”
- For RRA settings, see “WebSphere relational resource adapter settings” on page 505
- For information about CMP connection factories, see “Connection factory” on page 510
- For information about enterprise beans, see ″Introduction: EJB applications″ in the information center.

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);

    // Note: 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.  However, certain applications may need to differentiate
    // between a custom helper and an existing helper of the same type,
    // so WebSpehere has provided the value 'DataStoreHelper.CUSTOM_HELPER'
    // for this purpose.  If this functionality is needed by your application
    // insert the following line into your code:
    // getMetaData().setHelperType(DataStoreHelper.CUSTOM_HELPER);


}


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 are the connection factories 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 DataSource 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 can not 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 DataSource for CMP* checkbox.

To view this administrative console page, click **Resources** >**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** > *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 that must interact with *relational* databases use JDBC providers for data access. Together, the JDBC provider and data source objects are functionally equivalent to the J2EE Connector Architecture (JCA) connection factory (which provides access to non-relational databases).

The WebSphere Application Server prerequisite Web site has a current list of supported providers. If your database is DB2, however, you can proceed directly to the topic Vendor-specific data sources minimum required settings to learn which DB2 JDBC provider is appropriate for your database configuration and application requirements. This document contains descriptions of the following providers, including the supported data source classes and their required properties:

- DB2 for zOS Local JDBC Provider (RRS), for use with the DB2 for OS/390 and z/OS Legacy JDBC driver
- DB2 Universal JDBC Driver Provider, for use with the DB2 Universal JDBC driver
- DB2 Universal JDBC Driver Provider (XA), also for use with the DB2 Universal JDBC driver

Note: For software requirements and interoperability information regarding these providers, see the topics “DB2 Universal JDBC Driver Support” and “Provider coexistence considerations” on page 513.

DB2 Universal JDBC Driver Support:

This article lists the support details for using the DB2 Universal JDBC Driver with WebSphere Application Server for z/OS.

WebSphere Application Server for z/OS supports the DB2 Universal JDBC Driver. The capabilities available depend on the DB2 Universal JDBC Driver that you installed as follows:

- The z/OS Application Connectivity to DB2 for z/OS feature that provides DB2 Universal JDBC Driver Type 4 connectivity. This DB2 Universal JDBC Drive can be invoked only as a type 4 driver for z/OS. As a type 4 driver, it uses a communication protocol to communicate requests from a z/OS application to a remote DB2 database.

  When you install and configure this driver for WebSphere Application Server for z/OS, it permits your applications to use JDBC or Container Managed Persistence (CMP) support to access backend DB2 databases (DB2 V7 and up) residing on z/OS at any location. All global transactions are handled as J2EE XA transactions.
- The DB2 Universal JDBC Driver in DB2 UDB for z/OS Version 8. This driver provides both Type 2 and Type 4 support.

Type 4 driver support uses a communication protocol to communicate requests from a z/OS application to a remote DB2 database. This driver supports using J2EE XA transaction processing to process global transactions.

Type 2 driver support uses local API protocol to communicate requests from a z/OS application to a target DB2 running on the same z/OS system image as the application. When the Type 2 driver is used under z/OS, the driver supports the use of z/OS Resource Recovery Services (RRS) to coordinate global transactions across multiple resource manages using 2-phase commit processing.

When you install and configure this version of the driver, your applications can use JDBC or CMP support to access backend DB2 databases (DB2 V7 and up). These databases can reside on the same z/OS system image, or on a different z/OS system image. depending on the driver type used. Type 2 driver handles all global transactions as RRS-coordinated global transactions.

- The DB2 Universal JDBC Driver Provider by APAR PQ80841 on DB2 UDB for OS/390 and z/OS Version 7. This version provides both driver Type 2 and driver Type 4 support.

  Type 4 driver support uses a communication protocol to communicate requests from a z/OS application to a remote DB2 database. This driver supports using J2EE XA transaction processing to process global transactions.

  Type 2 driver support uses local API protocol to communicate requests from a z/OS application to a target DB2 running on the same z/OS system image as the application. When the Type 2 driver is used under z/OS, the driver supports the use of z/OS Resource Recovery Services (RRS) to coordinate global transactions across multiple resource manages using 2-phase commit processing.

  When you install and configure this version of the driver, your applications can use JDBC or CMP support to access backend DB2 databases (DB2 V7 and up). These databases can reside on the same z/OS system image, or on a different z/OS system image, depending on the driver type used.

Provider coexistence considerations: Following are provider coexistence possibilities.

DB2 Legacy JDBC Providers and DB2 Universal JDBC Driver Providers

- Under WebSphere Application Server for z/OS, JDBC Provider definitions that use the Legacy DB2 for OS/390 and z/OS JDBC Driver (db2j2classes.zip) and JDBC Provider definitions that use the new DB2 Universal JDBC Driver (db2jcc.jar) must be carefully configured to ensure they never coexist on the same server. This is because some of the same class names are used in both drivers and these duplicate classes are functionally different. Adding jar files for the two types of drivers to the same CLASSPATH causes unpredictable results, since incorrect classes will be used for the provider whose CLASSPATH definition is added last.
- Carefully define the scope in which the two different types of driver providers are used. If you define one provider type under one scope (cell, node, or server), and the other provider type under another scope, separation is not ensured if the two scopes include the same server.

  For example, if you define a DB2 for zOS Local JDBC Provider (RRS) at a node level, then define a DB2 Universal JDBC Driver Provider at a server level where the server is in the same node, the provider definition at the node level is propagated down to the server level. As a result, a conflict occurs between the JDBC drivers used by the two providers.

  To help you understand which WebSphere Application Server for z/OS providers cannot coexist together, the providers are listed below under the DB2 JDBC driver that it uses:

  1. Providers that use the DB2 for z/OS Legacy JDBC Driver
     - DB2 for zOS Local JDBC Provider (RRS)
  2. Providers that use the DB2 Universal JDBC Driver
     - DB2 Universal JDBC Driver Provider
     - DB2 Universal JDBC Driver Provider (XA)
     - Cloudscape Network Server Using Universal JDBC Driver Provider

DB2 Drivers and Cloudscape

- The Cloudscape Network Server Using Universal JDBC Driver Provider uses an embedded copy of the DB2 Universal JDBC Driver that is shipped with Cloudscape. This provider is configured to automatically use the new level of the DB2 Universal JDBC Driver when installing the DB2 Version 7, Version 8, or standalone Type 4 version of the DB2 Universal JDBC Driver on the system where WebSphere Application Server for z/OS is configured.

  The Cloudscape Network Server Using Universal JDBC Driver Provider can coexist on the same server as the DB2 for Universal JDBC Driver Providers since it uses the same DB2 JDBC driver.
- The Cloudscape Network Server Using Universal JDBC Driver Provider cannot coexist in the same server as the DB2 for zOS Local JDBC Provider (RRS), because the Cloudscape provider uses the DB2 Universal JDBC Driver and the DB2 for zOS Local JDBC Provider (RRS) uses the DB2 for OS/390 and z/OS Legacy JDBC Driver. These drivers conflict with one another.

Data sources

Installed applications uses a *data source* to access the data from the database.

A data source is associated with a JDBC provider that supplies the specific JDBC driver implementation class. The data source represents the J2EE Connector Architecture (JCA) connection factory for the relational resource adapter. Application components use the data source to access connection instances to a specific database; a connection pool is associated with each data source.

You can create multiple data sources with different settings, and associate them with the same JDBC provider. (One reason to do this is to provide access to different databases.) JDBC providers that are supported by WebSphere Application Server are required 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.

  Note: In one case a connection pool data source does support two-phase commit transactions: when the JDBC provider is DB2 for z/OS Local JDBC provider (RRS).

  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 (Version 4.0) - 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 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, 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, using the Version 6.0 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 SYSOUT of the application servant region.

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 non pooled 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.
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).

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.

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**

  There are two options to ensure that both CMP and BMP beans share the same physical connection:

  – 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 BMP is not 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.

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.)

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.

Connections and transactions:

All connection usage occurs within the scope of either a global transaction or a local transaction containment (LTC) boundary.

Connection behavior depends on your current operating scope. This article discusses some of the common characteristics you see when using connections in one of the transaction scopes.

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 523 topic for more details.

JDBC AutoCommit behavior

All JDBC connections, when first obtained through a getConnection() call, contain the setting AutoCommit = TRUE by default.

- 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 resources: 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.`

Passing client information to a database: Some databases enable you to set client information on the database connections using a backend-specific proprietary connection API. For some databases (such as DB2) you can also set the client information as a data source property. WebSphere Application Server before Version 6.0 only enables setting the client information as a data source property. This capability is somewhat limited, however, because the client information cannot be dynamically changed on the data source or the connections obtained from that data source. Also, setting the client information on the data source causes all connections created from that data source to have the same information. For example, if you set the ApplicationName as part of the data source *clientInformation*, all connections from that data source have the same application name. Because many different applications can access the same data source, this might not be desired.

With WebSphere Application Server Version 6.0, you can set the client information on some connections and not others, and you can set different client information on different database connections from the same data source. You can pass client information in one of two ways:

- Explicitly, by calling a proprietary API, *setClientInformation(Properties)* on the com.ibm.websphere.rsadapter.WSConnection.
- Implicitly, using the *WAS.clientinfo* trace string. You can enable this dynamically from the administrative console just like a normal trace. For more information about passing client information implicitly, see “Implicitly set client information” on page 531.

The API is defined on the *WSConnection* class which is part of the *com.ibm.websphere.rsadapter* package. You must cast your database connection in your applications to *com.ibm.websphere.rsadapter.WSConnection* before calling the API, as this is a WebSphere Application Server proprietary API. The API takes a properties object as an input parameter that provides the flexibility of adding new client information if and when it is introduced by the backend database, without any changes to this API itself.

```
public void setClientInformation (Properties props)throws SQLException;
```

For an example of using this API, 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.

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” on page 532 or “WAS.clientinfopluslogging trace” on page 532.

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,
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 thread ID, 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 WebSphere global security.

thread ID
An eight-character hexadecimal value that identifies the Java thread that controlls the processing of the application request within WebSphere Application Server. This ID is displayed in the trace header.

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 WebSphere 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 WebSphere 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).

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 1431 for more information.

Resource adapter archive file

A Resource Adapter Archive (RAR) file is a Java archive (JAR) file used to package a resource adapter for the Java 2 Connector (J2C) Architecture for WebSphere Application Server.

A RAR file can contain the following:

- Enterprise information system (EIS) supplied resource adapter implementation code in the form of JAR files or other runnable components, such as dynamic link lists.
- Utility classes.
- Static documents, such as HTML files, images, and sound files.

The standard file extension of a RAR file is *.rar*.

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
- CMP persistence functions
- Container-managed relationships
- Resource references
- 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
- Oracle
- DB2 database software
- 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)

CMP persistence functions

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

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 references

Though this article addresses the EJB 2.0 specification, you still might find parts of it pertinent to your environment.
- Accessing Databases from Web Applications

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 (Still pertinent to WebSphere Application Server Version 6.0)
- Database authentication in WebSphere Application Server V5 (Still pertinent to WebSphere Application Server Version 6.0)
- 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
- developerWorks article: Cloudscape Network Server with WebSphere Application Server

Oracle
- Oracle

DB2 database software
- DB2

Supported hardware, software, and APIs
- Supported hardware, software, and APIs

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.

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 prepared statements that are cached per connection. (A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently run the given SQL statement multiple times.) 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, 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 prepared statements that are cached on a given connection over the life of the server. Additional information about this setting...

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. Additional information about this and related settings...

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. Additional information about this and related settings...

Default: 1

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.

Tuning parameters vary according to the type of database you are using. ″DB2 tuning tips for z/OS″ are provided for your convenience. Because DB2 is not a WebSphere Application Server product and can change, consider these descriptions as suggestions.

Configuring data access with scripting

This topic contains the following tasks:

- “Configuring new J2C activation specs using scripting” on page 549
- “Testing data source connections using scripting” on page 552

Configuring a JDBC provider using 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 JDBC provider:

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 JDBCProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('JDBCProvider')
     ```

   Example output:

   ```
   Attribute     Type
   name     String
   implementationClassName   String
   ```

3. 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']]
     ```

4. 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)
   ```

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.

Attention: 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

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:

1. Identify the parent ID:
   - 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:
   - Using Jacl:
     ```
     $AdminConfig required DataSource
     ```
   - Using Jython:
     ```
     print AdminConfig.required('DataSource')
     ```

   Example output:
   ```
   Attribute   Type
   name    String
   ```
3. Setting up required attributes:
   - Using Jacl:
     ```
     set name [list name DS1]
     set dsAttrs [list $name]
     ```
   - Using Jython:
     ```
     name = ['name', 'DS1']
     dsAttrs = [name]
     ```
4. Create a data source:
   - Using Jacl:
     ```
     set newds [$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)
   ```
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 connection pools using 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 connection pool:

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.

Configuring new data source custom properties using 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:g

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

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

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 data source:

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

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

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

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 /currentScript/cicseci.rar
     set option  [list  -rar.name  RAR1]
     ```
   - Using Jython:
     ```
     rarFile = '/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

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

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

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 specs using 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 J2C activation specs:

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 specs defined, it is required that you specify the activation spec attribute. If your resource adapter is JCA V1.5 and you have only one activation spec 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 specs 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 spec 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

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

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}} /sqllib/java/db2java.zip en US"
     ```

   - Using Jython:

     ```
     print AdminControl.invoke(dshelper, 'testConnectionToDataSource',  'COM.ibm.db2.jdbc.DB2XADataSource
      dbuser1 dbpwd1 "{{databaseName jtest1}}" /sqllib/java12/db \"\" \"\"\"')
     ```

   Example output:

   ```
   WASX7217I: Connection to provided data source was successful.
   ```

Deploying data access applications

Before installing a data access application into the WebSphere Application Server environment, you must first ensure that the appropriate database objects are available. This action includes creating and configuring any databases or tables required, setting necessary configuration parameters to handle expected load, and configuring any necessary JDBC providers and data source objects for servlets, enterprise beans, and client applications to use.

1. If your database configuration does not already exist:
   a. Create a database to hold the data.
   b. Create tables required by your application.
      If your application uses entity enterprise 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 does not use entity beans
      You must use your database server interfaces to create the tables.
   c. See Minimum required properties for vendor-specific data sources for certain vendors' database requirements.
2. If your enterprise application contains a Web application or an EJB application that uses connection pooling to access a relational database, see "Creating and configuring a JDBC provider and data source" on page 561.

3. If your application requires access to a non-relational database, you need to configure a resource adapter and a connection factory rather than a JDBC provider and a data source.
4. If your enterprise application contains an application client that accesses a relational database, see Configuring data access for application clients.
5. Consider the security of lookups with component managed authentication. See Security of lookups with component managed authentication for more information.

Relationship of assembly and administrative console data access settings

This article provides miscellaneous tips for using supported databases. See also the related links.

Always consult the product documentation for a list of the database brands and versions that are supported by your particular WebSphere Application Server version, edition, and FixPak.

Notes about various databases

- When using local DB2 databases for data access by session clients on AIX Version 4.3.3 or later versions, in some cases you cannot establish multiple connections for session clients. This is because 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
    ```
- When using Sybase 11.x, you might encounter the following error when HttpSession persistence is enabled:

  ```
  DBPortability W Could not create database table: "sessions"
  com.sybase.jdbc2.jdbc.SybSQLException: The 'CREATE TABLE' command is not
  allowed within a multi-statement transaction in the 'database_name' database
  ```

  where *database_name* is the name of the database for holding sessions.

  If you encounter the error, issue the following commands at the Sybase command line:

  ```
  use database_name
  go
  sp_dboption db,"ddl in tran ",true
  go
  ```
- Sybase 12.0 does not support local transaction modes with a JTA enabled data source. To use a connection from a JTA enabled data source in a local transaction, install Sybase patch EBF9422.

Additional administrative tasks for specific databases

For your convenience, this article provides instructions for enabling some popular database drivers, and performing other administrative tasks often required to provide data access to applications running on WebSphere Application Server. These tasks are performed outside of the WebSphere Application Server administrative tools, often using the database product tools. Always refer to the documentation accompanying your database driver as the authoritative and complete source of driver information.

See the Supported hardware, software, and APIs for the latest information about supported databases, drivers, and operating systems.

- **Enabling JDBC 2.0**

  Ensure that your operating system environment is set up to support JDBC 2.0. This action is required to use data sources created through WebSphere Application Server.

The following steps make it possible to find the appropriate JDBC 2.0 driver for use with WebSphere Application Server administration:

- **Enabling JDBC 2.0 with DB2 on Windows NT systems**

  To enable JDBC 2.0 use on Windows NT systems:
  - For DB2 Version 7.2
    1. Stop the DB2 JDBC Applet Server service.
    2. Run the following batch file:

       ```
       SQLLIB\java12\usejdbc2.bat
       ```
    3. Stop WebSphere Application Server (if it is running) and start it again.
  - For DB2 Version 8.1
    - JDBC 2.0 is supported by default, there are no additional steps for you to perform.

  Perform the steps once for each system.

- **Determining the level of the JDBC API in use for DB2 on Windows NT systems**

  To determine the JDBC level in use on your system:
  - For DB2 Version 7.2
    - If JDBC 2.0 is in use, this file exists:

      ```
      SQLLIB\java12\inuse
      ```
    - If JDBC 1.0 is in use, this file exists:

      ```
      SQLLIB\java11\inuse
      ```

      or no *java11* directory exists.
  - For DB2 Version 8.1
    - Go to directory *SQLLIB\samples\java*, compile and run the class *db2JDBCVersion.java*.

- **Enabling JDBC 2.0 with DB2 on UNIX systems**
  - For DB2 Version 7.2
    - Before starting WebSphere Application Server, call *$INSTHOME/sqllib/java12/usejdbc2* to use JDBC 2.0. For convenience, you might want to put this in your root user's startup script. For example, on AIX, add the following to the root user's *.profile*:

      ```
      if [ -f /usr/lpp/db2_07_01/java12/usejdbc2 ] ; then
          . /usr/lpp/db2_07_01/java12/usejdbc2
      fi
      ```
  - For DB2 Version 8.1
    - JDBC 2.0 is supported by default, there are no additional steps for you to perform.

- **Determining the level of the JDBC API in use for DB2 on UNIX systems**
  - For DB2 Version 7.2
    - To determine if you are using JDBC 2.0, you can echo *$CLASSPATH*. If it contains

      ```
      $INSTHOME/sqllib/java12/db2java.zip
      ```

      then JDBC 2.0 is in use.

      If it contains

      ```
      $INSTHOME/sqllib/java/db2java.zip
      ```

      then JDBC 1.0 is in use.
  - For DB2 Version 8.1
    - Go to directory *sqllib/samples/java*, compile and run the class *db2JDBCVersion.java*.

- **Sourcing the db2profile script on UNIX systems**

  Before starting WebSphere Application Server to host applications requiring data access, source the *db2profile*:

  ```
  . ~db2inst1/sqllib/db2profile
  ```

  where *db2inst1* is the user created during DB2 installation.

- **Using Java Transaction API drivers**

  Instructions are available for using Java Transaction API (JTA) drivers on particular operating systems. See your operating system documentation for more information.

The goal of this section is to provide information about the steps that make DB2 work well with applications utilizing XA classes -- that is, those whose *dataSourceClasses* implement *javax.sql.XADataSource*.

- **Using Java Transaction API drivers for DB2 on Windows NT systems**

  To enable JTA drivers for DB2 on Windows NT systems, follow 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`

- **Using Java Transaction API drivers for DB2 on UNIX systems**

  To enable JTA drivers on UNIX systems, follow 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`

- **For Oracle 8.1.7 two phase commit support**

  You can use the Oracle 8.1.7 thin driver for JTA two-phase support with the following restrictions:

  - The thin driver that comes shipped with 8.1.7 might or might not work. Future patches from Oracle might work as well, but are not tested. The driver that was available from the Oracle Technology Network Web site as of February 20, 2001 does work and is the recommended driver. Later versions on this Web site are expected to work, but are not tested.

    To obtain the driver from the Oracle support Web site, visit:

    `http://technet.oracle.com/`

    You need to be a registered user for the Oracle Technology Network to get the driver from this site. Contact Oracle for access. After you have access download the 8.1.7 driver for the platforms you use and follow the instructions for installing the new driver.
  - You must use the 8.1.7 driver with 8.1.7 databases, 8.1.6 databases do not support the recover() and forget() methods and other problems are encountered running with 8.1.6. Oracle does not support JTA with 8.1.6.
  - For Oracle, you can only use JTA with container-managed persistence (CMP) beans.
  - For the bean to create the table, you must start the bean with the JTA set to *false*. After the bean creates the table, you can set the JTA back to *true*.
  - Configure an entity bean that accesses Oracle with JTA set to *true* as follows:

- Click **deployment descriptor properties** > **Transactions** > Remote tab. Set the Transaction Attribute to *TX_REQUIRED*.
- Click **Isolation** > Remote tab. Set the Isolation Level to *TRANSACTION_READ_COMMITTED*.

– Configure a session bean that is used with an entity bean that accesses Oracle with JTA set to *true* as follows:
- Click **deployment descriptor properties** > **Transactions** > Remote tab. Set the Transaction Attribute to *TX_BEAN_MANAGED*.
- Click **Isolation** > Remote tab. Set the Isolation Level to *TRANSACTION_READ_COMMITTED*.

- **Using Java Transaction API drivers for Sybase products on AIX systems**

  To enable Java Transaction API (JTA) drivers for use with Sybase products on the AIX operating system, follow 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
     ```

- **Notes about Sybase Java Transaction API drivers**

  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.

  Related concepts

  "Data sources" on page 514
  Installed applications uses a *data source* to access the data from the database.

  Related reference

  "Data access : Resources for learning" on page 533

Recreating database tables from the exported table data definition language:

When the WebSphere Application Server deployment tooling deploys an EJB jar file containing container-managed persistence (CMP) enterprise beans, it selects the target database and creates a corresponding `Table.ddl` file. This file contains the SQL statement necessary to generate the database table for your CMP beans. You must then run the `ddl` file on your database server to create the tables.

Following is an example of how to use such a `Table.ddl` file for DB2, on Windows and z/OS:

1. To create your tables using SPUFI, extract the `Table.ddl` file located in your EJB JAR, and save it to a temporary directory on your work station.
2. Transfer this `Table.ddl` file to a data set on your z/OS system.
3. Specify the data set as the input data set to SPUFI.

Installing J2EE Connector resource adapters

1. Click **Resources**.
2. Click **Resource Adapters**.
3. Select the *scope* at which you want to define this resource adapter. (This scope becomes the scope of your connection factory.) You can choose cell, node, cluster, or server. For more information, see ″Administrative console scope settings″.

4. Click **Install RAR**. The Install RAR button opens a dialog that enables you to install a J2EE Connector Architecture (JCA) connector and create a resource adapter for it. You can also use the **New** button, but the New button creates only a new resource adapter (the JCA connector must already be installed on the system).

   Note: 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. The node on which the file is installed is determined by the scope on the **Install RAR** page. (The scope you set on the Resource Adapters page determines the scope of the new resource adapters, which you can install at the server, node, or cell level.)
5. 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.
6. Click **Next**.
7. Enter the resource adapter name and any other properties needed under *General Properties*. 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.
8. 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 559 and “WebSphere relational resource adapter settings” on page 505
3. Click **Finish** > **Save** to save the changes.
4. Create connection factories for the newly installed application.
   a. Open the administrative console.
   b. Click **Applications** > **Enterprise Applications** > *application name*.
   c. Click **Connector Modules** in the Related Items section of the page.
   d. Click *filename.rar*.
   e. Click **Resource adapter** in the Additional Properties section of the page.
   f. Click **J2C Connection Factories** in the Additional Properties section of the page.
   g. Click on an existing connection factory to update it, or **New** to create a new one.

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.

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**.

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**.

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

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_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.

Data type String

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.)

Creating and configuring a JDBC provider and data source

This topic outlines the process for configuring access to a relational database, and provides links to more detailed instruction.

Determine which version of data source you need. If you are using the Enterprise JavaBean (EJB) 1.0 specification or the Java Servlet 2.2 specification, you need the Version 4.0 data source. If you are using more advanced releases of these specifications, you need the current version data source (which is designated in WebSphere Application Server simply as ″Data source,″ with no associated version number).

Next, determine which JDBC provider is appropriate for your data source version, database configuration, and application requirements. The topic “Vendor-specific data sources minimum required settings” is a comprehensive source for this information. It includes a summary table and detailed listings.

1. 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 537.

   OR

   Using the Java Management Extensions (JMX) API, see Creating a JDBC provider and data source using the Java Management Extensions API.
2. 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 538. (For V4 data sources, see “Configuring new WAS40 data sources using scripting” on page 541.)

   OR

   Using the JMX API, see Creating a JDBC provider and data source using the Java Management Extensions API.
3. Bind the resource reference. See ″Binding to a data source″ in the information center.
4. Test the connection (for non-container-managed persistence usage). See Test connection.

   Note: When you save the data source configuration, it is saved in a *resource.xml* file. In most environments, you should not need to modify the **jdbc-resource-provider-templates.xml** file. However, if updating the file becomes necessary, observe the special considerations that are discussed in ″J2EE Connector Architecture migration tips″.

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	One phase only	
	DB2 legacy CLI-based Type 2 JDBC Provider (XA)	One and two phase	
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	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)	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)	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	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)	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)
Cloudscape	Cloudscape JDBC Provider	One phase only	**-** Not for use in clustering environment: Cloudscape is accessible from a single JVM only **-** Does not support Version 4 data sources
	Cloudscape JDBC Provider (XA)	One and two phase	**-** Not for use in clustering environment: Cloudscape is accessible from a single JVM only **-** Does not support Version 4 data sources
	Cloudscape Network Server using Universal JDBC driver	One phase only	Does not support Version 4 data sources
Informix	Informix JDBC Driver	One phase only	
	Informix JDBC Driver (XA)	One and two phase	
Sybase	Sybase JDBC Driver	One phase only	
	Sybase JDBC Driver (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
Oracle	Oracle JDBC Driver	One phase only	
	Oracle JDBC Driver (XA)	One and two phase	
MS SQL Server	DataDirect ConnectJDBC type 4 driver for MS SQL Server	One phase only	Only for use with the corresponding driver from DataDirect Technologies
	DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA)	One and two phase	Only for use with the corresponding driver from DataDirect Technologies
	WebSphere embedded ConnectJDBC driver for MS SQL Server	One phase only	- Not available for Application Server on z/OS - Cannot be used outside of WebSphere Application Server environment
	WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)	One and two phase	- Not available for Application Server on z/OS - Cannot be used outside of WebSphere Application Server environment

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 Supported hardware and software Web page at the http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html IBM Web address.

Use these links to find your provider and data source information:
- DB2 on z/OS, connecting to Application Server on z/OS
- “DB2 UDB for iSeries, connecting to Application Server on z/OS” on page 567
- Cloudscape
- Informix
- Sybase
- Oracle
- MS SQL Server

DB2 on z/OS, connecting to Application Server on z/OS

1. **DB2 for zOS Local JDBC Provider (RRS)**

   The DB2 for zOS Local JDBC Provider (RRS) is for use with the DB2 for 390 and z/OS Legacy JDBC Driver. It can be used only with WebSphere Application Server for z/OS. This provider supports the creation of WebSphere Application Server for z/OS v5.0 and v4.0 data sources. It also uses z/OS Resource Recovery Services (RRS) to coordinate transactions across multiple resource managers using two-phase commit processing.

The DB2 for zOS Local JDBC Provider (RRS) allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access to DB2 databases. Use of SQLJ with Container Managed Persistence (CMP) is not supported under this provider.

To use this provider, the legacy DB2 for z/OS JDBC Driver must be installed and configured to the WebSphere Application Server. Refer to the topic "Using a DB2 for z/OS Local JDBC Provider (RRS) with WebSphere Application Server for z/OS" on page 582.

The following configuration information is provided in a template for the DB2 for zOS Local JDBC Driver Provider (RRS), and is automatically filled in when you select this provider.

Data source implementation:

`com.ibm.db2.jcc.DB2ConnectionPoolDataSource`

This DB2 data source allows WebSphere Application Server for z/OS to perform connection pooling. Note that when you configure the data source, you must specify a name for the data source definition.

This provider requires JDBC driver files:

- `db2j2classes.zip`, which are DB2 for z/OS Legacy JDBC driver files. They can be retrieved using the following class path:

  `${DB2390_JDBC_DRIVER_PATH}/classes/db2j2classes.zip`
- The native files (`.so` type files) required by the DB2 for OS/390 and z/OS Legacy JDBC driver, retrievable from the following library path:

  `${DB2390_JDBC_DRIVER_PATH}/lib`

The driver requires DataStoreHelper class:

`com.ibm.websphere.rsadapter.DB2DataStoreHelper`

It also requires a valid authentication alias. When res-auth = CONTAINER is used, however, it is permissible to not specify any authentication alias. In this case, the user identity associated with a connection created by the data source is the user identity associated with the current thread at the time a connection request is made.

This driver requires the following properties:

- **databaseName** The location name of the target database, used when establishing connections using this data source.

2. **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.

   Note: To use this provider, you must have the DB2 Universal JDBC Driver for DB2 Version 7 or DB2 Version 8 installed and configured for WebSphere Application Server for z/OS. Refer to the topic "Using a DB2 Universal JDBC Driver Provider with WebSphere Application Server for z/OS" on page 585.

   The DB2 Universal JDBC Provider allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access to DB2 databases. SQLJ use with CMP is also supported.

   It supports the following data source:

   `com.ibm.db2.jcc.DB2ConnectionPoolDataSource`

   Note that when you configure the data source, you must specify a name for the data source definition. This data source implementation class performs one-phase commit processing when you specify driver Type 4 in WebSphere Application Server for z/OS. When you specify driver Type 2, Application Server for z/OS uses RRS to coordinate transaction processing, and two-phase commit processing is performed for global transactions. The provider requires JDBC driver files:

   - `db2jcc.jar` This is the DB2 Universal JDBC Driver jar file. After the DB2 installation, this jar file is located in DB2's install directory. The fully-qualified path of this jar must be specified as the value of the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable. Class path:

     `${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar`

- db2jcc_License_cu.jar This is the DB2 Universal JDBC driver license file that allows access to DB2 Universal databases under Cloudscape and workstations. It is not used for WebSphere Application Server for z/OS, but is included to make the provider definition common between WebSphere Application Server for z/OS and WebSphere Application Server Distributed. 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 DB2 Universal databases under Cloudscape, workstations, and z/OS. After you install DB2, this jar file appears in the same DB2 directory as db2jcc.jar. Class path:

  ```
  ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cisuz.jar
  ```
- The native files (.so type files) required by the DB2 Universal JDBC Driver in WebSphere Application Server for z/OS. Use the following library path:

  ```
  ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}
  ```

The DB2 Universal JDBC driver requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
```

This driver also requires a valid authentication alias if the driverType property (see properties below) is set to 4. If the driverType property is set to 2, a component-managed authentication alias must be specified to use the datasource with res-auth APPLICATION. In the case where driverType 2 is specified and the datasource is used with res-auth CONTAINER, you can specify a container-managed authentication alias; however, it is not required. If you do not specify a container-managed alias, the user identity associated with a connection created by the datasource will be the user identity associated with the current thread at the time the connection is obtained.

It requires the following 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**.

3. **DB2 Universal JDBC Provider (XA)**

   This provider is the XA DB2 Universal JDBC provider that uses the DB2 Universal JDBC driver to provide access to DB2 databases. The Universal JDBC driver supports Java communication-based connectivity (driver Type 4), which allows distributed access to DB2. The driver also supports Java Native Interface (JNI) based connectivity (driver type 2), which allows local access to DB2. For XA capabilities, however, driver type 2 is not supported by the DB2 Universal JDBC Driver on WebSphere Application Server for z/OS. Therefore driver type 2 should *not* be used when defining an XA data source under this provider.

   Note: To use this provider, you must have the DB2 Universal JDBC Driver for DB2 Version 7 or DB2 Version 8 installed and configured for WebSphere Application Server for z/OS, or you must have the z/OS Application Connectivity to DB2 for z/OS feature installed and configured for WebSphere Application Server for z/OS. Refer to the topic “Using a DB2 Universal JDBC Driver Provider with WebSphere Application Server for z/OS” on page 585.

   The DB2 Universal JDBC Provider (XA) allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access to DB2 databases. SQLJ use with CMP is also supported.

   This provider does not support the creation of Version 4.0 data sources.

   The DB2 Universal JDBC Provider (XA) supports the two phase data source:

   com.ibm.db2.jcc.DB2XADataSource

   Note that when you configure the data source, you must specify a name for the data source definition.

It requires JDBC driver files:

- `db2jcc.jar` This is the DB2 Universal JDBC Driver jar file. After the DB2 installation, this jar file is located in DB2's install directory. The fully-qualified path of this jar must be specified 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 DB2 Universal databases under Cloudscape and workstations. It is not used for WebSphere Application Server for z/OS, but is included to make the provider definition common between WebSphere Application Server for z/OS and WebSphere Application Server Distributed. 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 DB2 Universal databases under Cloudscape, workstations, and z/OS. After you install DB2, this jar file appears in the same DB2 directory as db2jcc.jar. Class path:

  `${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cisuz.jar`
- The native files (`.so` type files) required by the DB2 Universal JDBC Driver in WebSphere Application Server for z/OS. Use the following library path:

  `${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}`

  (In cases that do not require native files, set the DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH to null.)

The driver requires **`DataStoreHelper`** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

It also requires a valid authentication alias.

The DB2 Universal JDBC driver requires the following 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**.

For more information on DB2 for z/OS, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

DB2 UDB for iSeries, connecting to Application Server on z/OS

In the rare case that you need to connect to DB2 UDB on iSeries to provide JDBC connectivity for an application run on WebSphere Application Server for z/OS, you can use the iSeries Toolbox driver for Java, the iSeries Toolbox XA-compliant driver for Java, or the DB2 JDBC Universal Driver XA.

1. **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`

   Requires an authentication alias.

   Requires properties:

- **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

2. **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`

   Requires an authentication alias.

   Requires properties:
   - **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

3. **DB2 Universal JDBC Provider (XA)**

   This provider is the XA DB2 Universal JDBC provider that uses the DB2 Universal JDBC driver to provide access to DB2 databases. The Universal JDBC driver supports Java communication-based connectivity (driver Type 4), which allows distributed access to DB2. If you are running WebSphere Application Server for z/OS and connecting to DB2 UDB for iSeries, you *cannot* use Java Native Interface (JNI) based connectivity (driver type 2) with this provider.

   Note: To use this provider, you must have the DB2 Universal JDBC Driver for DB2 Version 7 or DB2 Version 8 installed and configured for WebSphere Application Server for z/OS, or you must have the z/OS Application Connectivity to DB2 for z/OS feature installed and configured for WebSphere Application Server for z/OS. Refer to the topic “Using a DB2 Universal JDBC Driver Provider with WebSphere Application Server for z/OS” on page 585.

   The DB2 Universal JDBC Provider (XA) allows applications to use both JDBC and Structured Query Language in Java (SQLJ) access to DB2 databases. SQLJ use with CMP is also supported.

   This provider does not support the creation of Version 4.0 data sources.

   The DB2 Universal JDBC Provider (XA) supports the two phase data source:

   com.ibm.db2.jcc.DB2XADataSource

   Note that when you configure the data source, you must specify a name for the data source definition.

   It requires JDBC driver files:
   - `db2jcc.jar` This is the DB2 Universal JDBC Driver jar file. After the DB2 installation, this jar file is located in DB2’s install directory. The fully-qualified path of this jar must be specified 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 DB2 Universal databases under Cloudscape and workstations. It is not used for WebSphere Application Server for z/OS, but is included to make the provider definition common between WebSphere Application Server for z/OS and WebSphere Application Server Distributed. 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 DB2 Universal databases under Cloudscape, workstations, and z/OS. After you install DB2, this jar file appears in the same DB2 directory as db2jcc.jar. Class path:

     `${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_License_cisuz.jar`
   - The native files (`.so` type files) required by the DB2 Universal JDBC Driver in WebSphere Application Server for z/OS. Use the following library path:

     `${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}`

(In cases that do not require native files, set the DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH to null.)

The driver requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper`

It also requires a valid authentication alias.

The DB2 Universal JDBC driver requires the following properties:

- **databaseName** This is a locally cataloged database name.
- **driverType** This is the JDBC connectivity type of a data source. Set this property to type 4 if you are running WebSphere Application Server for z/OS and connecting to DB2 UDB for iSeries.
- **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**.

For more information on DB2 for iSeries, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/.

Cloudscape

1. **Cloudscape JDBC Provider**

   The Cloudscape JDBC Provider provides the JDBC access to the Cloudscape database. This Cloudscape JDBC driver used the embedded framework. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape JDBC Provider supports one phase data source:

   com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource

   Requires JDBC driver files: **db2j.jar.**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

   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./cloudscape` (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. **Cloudscape JDBC Provider (XA)**

   The Cloudscape JDBC Provider (XA) provides the XA-compliant JDBC access to the Cloudscape database. This Cloudscape JDBC driver uses the embedded framework. You cannot use any Version 4.0 data sources with Cloudscape.

   Cloudscape JDBC Provider (XA) supports two phase data source:

   com.ibm.db2j.jdbc.DB2jXADataSource

   Requires JDBC driver files: **db2j.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper`

   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./cloudscape` (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. **Cloudscape Network Server using Universal JDBC driver**

This Cloudscape 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.

Cloudscape uses the DB2 Universal Driver when using the Network Server. It supports one phase data source:

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 Cloudscape network server is set to be:

```
<classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar</classpath>

<classpath>${CLOUDSCAPE_JDBC_DRIVER_PATH}/db2j.jar</classpath>

<classpath>${CLOUDSCAPE51_JDBC_DRIVER_PATH}/db2j.jar</classpath>

<classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.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 when you set up your DB2 datasource. This is instead of hard coding the path of the db2jcc.jar for DB2 datasources.
- **db2jcc_license_cu.jar** This file is the DB2 Universal JDBC license file that provides access to the Cloudscape 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 defined to the location in Websphere Application Server where the license jar file above 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: Cloudscape requires only db2jcc_license_c.jar; however, WebSphere Application Server uses db2jcc_license_cu.jar because this works for both DB2 UDB and Cloudscape.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper`

Note: The administrative console incorrectly lists the DB2UniversalDataStoreHelper as the default value for the **DataStoreHelper** class. You must change the default value to `com.ibm.websphere.rsadapter.CloudscapeNetworkServerDataStoreHelper`. Also change the custom properties, using the instructions in the customer property section.

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./cloudscape` (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.

See the Cloudscape setup instructions for more information on configuring the Cloudscape Network Server.

For more information on IBM Cloudscape, visit the Cloudscape Web site at: http://www.ibm.com/software/data/cloudscape/

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*.
   - **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`

   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*.
- **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*.

For more information on Informix, visit the Informix Web site at: http://www.ibm.com/software/data/informix/

Sybase

1. **Sybase JDBC Driver**

   The Sybase JDBC Driver is a Type 4 JDBC driver that provides JDBC access to the Sybase database.

   Sybase JDBC Driver supports one phase data source:

   com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource

   Requires JDBC driver files: **jconn2.jar**.

   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 JDBC Driver (XA)**

   The Sybase JDBC Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Sybase database.

   Sybase JDBC Driver (XA) supports two phase data source:

   com.sybase.jdbc2.jdbc.SybXADataSource

   Requires JDBC driver files: **jconn2.jar**.

   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`

   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

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 database. 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 database. 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. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   DataDirect SequeLink type 3 JDBC driver for MS SQL Server is a type 3 JDBC driver that provides JDBC access to MS SQL Server via SequeLink server.

   This JDBC provider supports this 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*.

4. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server (XA)** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   DataDirect SequeLink type 3 JDBC driver for MS SQL Server (XA) is a type 3 JDBC driver that provides XA-compliant JDBC access to MS SQL Server via the SequeLink server.

This JDBC provider supports this 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*.

Both of the WebSphere-embedded SequeLink JDBC drivers require 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. (Only install SequeLink Server from the WebSphere Application Server CD if you are using the SequeLink JDBC driver embedded in WebSphere. Otherwise, install a copy of SequeLink Server purchased from DataDirect Technologies.)

From the following FTP site, you can download the latest patches and upgrades for the version of SequeLink Server that is used with the WebSphere-embedded SequeLink JDBC driver:

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/

5. **Microsoft JDBC driver for MSSQLServer 2000** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   Microsoft JDBC driver for MSSQLServer 2000 is a type 4 JDBC driver that provides JDBC access to the MS SQL Server database.

   This JDBC provider supports this 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*.

6. **Microsoft JDBC driver for MSSQLServer 2000 (XA)** -- Deprecated

   Because this JDBC provider is deprecated in WebSphere Application Server Version 6.0, it is no longer an available option in the administrative console. In its place, use one of the Connect JDBC providers, which are described previously in this section.

   Microsoft JDBC driver for MSSQLServer 2000 (XA) is a type 4 JDBC driver that provides XA-compliant JDBC access to the MS SQL Server database.

   This JDBC provider supports this 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*.

For more information on the Microsoft JDBC driver, visit the Microsoft Web site at:

http://www.microsoft.com/sql

Creating and configuring a JDBC provider using the administrative console:

An application installed on WebSphere Application Server accesses a relational database through a JDBC provider, which is essentially a system-level software driver. For at-a-glance information on which provider type is appropriate for your database configuration and application requirements, see the JDBC provider table.

You can easily establish a JDBC provider from the administrative console.

1. Open the administrative console.
2. Click **Resources** > **JDBC Providers**.
3. Select the *scope* of your definition. (This scope becomes the scope of your data source.) You can choose cell, node, cluster, or server. For more information, see Administrative console scope settings.
4. Click **New**.

   Note: If Java script is disabled for your browser, you do not see the three drop-down lists that are described in the next three steps (for database type, provider type, and implementation type) . Instead, you see a single drop-down box that lists *all* JDBC provider choices simultaneously (inclusive of every database, provider, and implementation type).
5. Use the first drop-down list to select the database type of the JDBC provider you need to create. If the list of supported JDBC provider types does not include the JDBC provider that you want to use, select the **User-Defined JDBC Provider**. Then consult the JDBC provider vendor's documentation for information on specific properties required for data sources associated with this provider, and skip to step eight of this list.
6. From the second drop-down list, select your JDBC provider type.
7. From the third drop-down list, select the implementation type 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 using this data source configuration have the benefit of container-managed transaction recovery.

8. Click **Next** to view the general property settings page for your JDBC provider.
9. Ensure that all required properties have valid values. For more information, see JDBC Provider settings.
10. Click **Apply** to view the page with your new JDBC provider settings. Note that two active data source links now appear under the **Additional Properties** heading on this page. To set up a data source, click the link that corresponds to the type required by your application, the Version 4 data source or the later version data source. (For more information, refer to the section entitled ″Choice of data source″ in the “Data sources” on page 514 topic.)
11. Click **OK** to return to the JDBC providers page, where your new JDBC provider appears in the list.

    Attention: If you modify the class path or native library path of a JDBC provider: After clicking **OK**, 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.

For detailed information on creating a data source for association with your JDBC provider, see “Creating and configuring a data source using the administrative console” on page 589.

JDBC Provider collection:

Use this page to view a JDBC provider.

To view this administrative console page, click **Resources** > **JDBC Providers** in the console navigation tree.

Notice the *Scope* of your JDBC provider. If you pick anything other than the default of *Node* the provider might not be available in other scope contexts. New items created in this view are created within the selected scope.

Name:

Specifies a text identifier for this provider.

For example, this field can be *DB2 JDBC Provider (XA)*.

Data type String

Description:

Specifies a text string describing this provider.

Data type String

JDBC provider settings:

Use this page to create or modify JDBC provider settings.

To view this administrative console page, click **Resources** > **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.

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, */usr/lpp/db2/db2710/classes/db2j2classes.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

New JDBC Provider:

Use this page to create a new JDBC provider.

To view this administrative console page, click **Resources** >**JDBC Providers** > **New**.

Step 1: Select the database type:

Choose a supported database type.

If the list of supported database types does not include the type that you want to use, select **User-Defined**. You might need to consult the documentation for the database for more information on specific properties that database requires.

Data type Drop-down list

Step 2: Select the JDBC provider type:

Choose a supported JDBC Provider type.

Only JDBC provider types that are appropriate for the database type you selected in step 1 will appear in the list. For at-a-glance information on which provider type is appropriate for your database configuration and application requirements, see the JDBC provider table.

Data type Drop-down list

Step 3: Select the implementation type:

Choose a supported implementation type.

Only the implementation types supported by the JDBC provider you selected in step 2 will appear in the list.

Note: If two choices appear on the implementation type list, select **Connection Pool DataSource** if your application runs in a single phase or local transaction. Otherwise, choose **XA DataSource** to run in a global transaction.

Data type Drop-down list

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	One phase only	
	DB2 legacy CLI-based Type 2 JDBC Provider (XA)	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	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)	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)	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	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)	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)
Cloudscape	Cloudscape JDBC Provider	One phase only	**-** Not for use in clustering environment: Cloudscape is accessible from a single JVM only **-** Does not support Version 4 data sources
	Cloudscape JDBC Provider (XA)	One and two phase	**-** Not for use in clustering environment: Cloudscape is accessible from a single JVM only **-** Does not support Version 4 data sources
	Cloudscape Network Server using Universal JDBC driver	One phase only	Does not support Version 4 data sources
Informix	Informix JDBC Driver	One phase only	
	Informix JDBC Driver (XA)	One and two phase	
Sybase	Sybase JDBC Driver	One phase only	
	Sybase JDBC Driver (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
Oracle	Oracle JDBC Driver	One phase only	
	Oracle JDBC Driver (XA)	One and two phase	
MS SQL Server	DataDirect ConnectJDBC type 4 driver for MS SQL Server	One phase only	Only for use with the corresponding driver from DataDirect Technologies
	DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA)	One and two phase	Only for use with the corresponding driver from DataDirect Technologies
	WebSphere embedded ConnectJDBC driver for MS SQL Server	One phase only	- Not available for Application Server on z/OS - Cannot be used outside of WebSphere Application Server environment
	WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)	One and two phase	- Not available for Application Server on z/OS - Cannot be used outside of WebSphere Application Server environment

Using a DB2 for z/OS Local JDBC Provider (RRS) with WebSphere Application Server for z/OS:

Setting up a JDBC provider for DB2 for z/OS and creation of a data source in WebSphere Application Server for z/OS is complicated by the fact that the JDBC driver must locate and load the DSNJDBC_JDBCProfile.ser file. DB2 for z/OS Version 7 has enhanced the mechanism by which the JDBC driver locates and loads this file to accommodate WebSphere Application Server for z/OS.

The steps to install a JDBC provider using this new support are:
- Update the PROCs for the servant address spaces to include the DB2 libraries. (optional)
- Create/update the db2sqljjdbc.properties file to indicate the location of the DSNJDBC_JDBCProfile.ser file.
- Create/update WebSphere environment variables so the WebSphere server can locate the DB2 home directory and the JDBC driver can locate the db2sqljjdbc.properties file.
- Define J2C Authentication Aliases to provide userid with password to be used when connecting to DB2k.
- Define the DB2 for z/OS Local JDBC Provider (RRS) for data access.
- Define a data source specifying the aliases for component and container connections. (optional)

1. Update the PROCs for the servant address spaces to include the DB2 libraries. (optional)

   Installations have the option of placing the DB2 libraries in the linklist or //STEPLIB DD concatenation for the WebSphere address spaces that will use JDBC. In some installations a combination of techniques are used. If your installation does not have SDSNEXIT, SDSNLOAD and SDSNLOD2 in the linklist, then you must update the //STEPLIB DD concatenation for the servant address space with the missing libraries.

   Consider the following example:

   The system on which WebSphere for z/OS Version 5 is active has multiple DB2 subsystems of different versions. There are the JUDY and DBP3 subsystems, where DBP3 is the default DB2 subsystem from a linklist perspective. The p5serv1 server uses the JUDY subsystem, hence the servant address space associated with the p5srv1 server must specify the appropriate DB2 libraries in the //STEPLIB DD concatenation.

On this system, the servant address space uses procedure P5NCASR, which includes member P5NCASRZ, which contains the //STEPLIB DD concatenation. The last three lines in the following are the updates, in gray area, which that were made for JDBC access to the JUDY subsystem.

```
 //* P5NCASRZ
//*
//* Output DDs
//*
//CEEDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//*
//*Steplib Setup
//*
//STEPLIB DD DISP=SHR,DSN=WAS500.WAS.SBBOLD2
// DD DISP=SHR,DSN=WAS500.WAS.SBBOLOAD
// DD DISP=SHR,DSN=DSN710.JUDY.SDSNEXIT
// DD DISP=SHR,DSN=DSN710.SDSNLOAD
// DD DISP=SHR,DSN=DSN710.SDSNLOD2
//*
```

2. Create/update the db2sqljjdbc.properties file to indicate the location of the DSNJDBC_JDBCProfile.ser file.

   The db2sqljjdbc.properties file contains information which the DB2 JDBC driver uses to configure itself. The following are the contents of the db2sqljjdbc.properties file created for the use of the WebSphere servers running in node *nodec* of cell *p5cell* and in particular server *p5srv1*.

   The area near the bottom under db2.connpool.max.size=100 indicates the new means by which the location of the DSNJDBC_JDBCProfile.ser file is specified using as of APAR PQ69861.

```
# >>> Start of p5nc_wscjudy_db2sqljjdbc.properties <<<
#
# Any lines starting with the pound sign '#'
# are comments. Please see the DB2 for OS/390
# Application Programming Guide and Reference
# for Java for the description of these settings.
#
#
# This DBRMLIB is specific for the DSN7 subsystem and the JDBC driver.
#
#DB2SQLJDBRMLIB=DSN710.DBRMLIB.DATA
#
# This is installation specific: the name of the DB2 subsystem to use.
#
DB2SQLJSSID=JUDY
#
# The next 5 items are default values which are coded for documentation
#
#
DB2SQLJPLANNAME=DSNJDBC
DB2SQLJJDBCPROGRAM=DSNJDBC
DB2SQLJMULTICONTEXT=YES
DB2SQLJATTACHTYPE=RRSAF
DB2CURSORHOLD=YES
#
#
# The following items are for tracing, there are no default values
#
#DB2SQLJ_TRACE_FILENAME=/tmp/p5srv1_jdbc
#DB2SQLJ_TRACE_BUFFSIZE=256
#DB2SQLJ_TRACE_WRAP=1
#
#
db2.connpool.max.size=100
#
# The following allows the JDBC driver to find and
# load the serialized profile.
#
```

```
db2.jdbc.profile.pathname=/usr/lpp/db2710/classes/DSNJDBC_JDBCProfile.ser
#
# <<< End of p5nc_wscjudy_db2sqljjdbc.properties >>>
```

The key entries in this file are:

DB2SQLJSSID
Identifies the DB2 subsystem (for example, JUDY) that should be used by the JDBC driver

db2.jdbc.profile.pathname
Identifies the serialized profile (for example, /usr/lpp/db2710/classes/DNSJDBC_JDBCProfile.ser) to be used by the JDBC dsriver. This is a new property introduced by APAR PQ69861.

Note: WebSphere for z/OS has always recommended specifying the db2sqljjdbc.properties and contents rather than assuming the defaults. While it is possible for the JDBC driver to locate the DB2 subsystem through specifying SSID in the DSNHDECP module, which resides in SDSNEXT library, rather than db2sqljjdbc.properties file, it is useful having a properties file for several reasons. Being able to specify the location of the serialized profile makes the definition of the DB2 JDBC provider less complicated.

Note: This file must be placed in the HFS in a directory that is readable by the all the servant address spaces on the z/OS image. It must be in codepage Cp1047 (for example, EBCDIC).

3. Create/update WebSphere environment variables so the WebSphere server can locate the DB2 home directory and the JDBC driver can locate the db2sqljjdbc.properties file.

   The WebSphere for z/OS configuration needs to have WebSphere and POSIX environment variables set so that the DB2 home directory and the db2sqljjdbc.properties file can be located. Click on Environment > Manage WebSphere Variables. Select the node level view and click Apply.

   There are two variables of interest.

   a. **DB2390_JDBC_DRIVER_PATH** This variable will be present but have no assigned value. Update this value to the directory in which the DB2 code resides. For example: /usr/lpp/db2/db2710
   b. **DB2SQLJPROPERTIES** This variable will not be present. Add this value to indicate the location and name of DB2 JDBCSQLJ properties file which you created for this node. For example: /shared/zWebSphere/V5/misc/p5nc_wscjudy_db2sqljjdbc.properties

   Note: These variables are set at the node level and thus are applicable to all servers in the node residing on this system. They can be set at the server level instead of or in addition to setting them at the node level.

   Note: The DB2390_JDBC_DRIVER_PATH variable could actually be set at the cell level and removed from the node level definitions if all nodes in the cell will use the same DB2 home directory.

   Note: The DB2SQLJPROPERTIES should not be set at the cell level, as it contains the specific DB2 subsystem to which the JDBC driver will use.

   Note: The /shared/zWebSphere/V5/misc/ directory is a commonly accessible directory for all systems in the sysplex. The name of the file indicates which cell (for example, p5) and node (for example, nc) that it is intended to be used. It is certainly possible to use a non-shared directory that is system specific (for example, /etc/zWebSphere/V5/misc/...)

   Note: After these updates have been made and saved, it is necessary to restart the server to have the server use the //STEPLIB DD updates from Step 1, if required, and to have the POSIX runtime environment variables refreshed with the changes made in this step. This restart can be delayed, but it must be done before the data source created in the last step can be used.

4. Define J2C Authentication Aliases to provide userid with password to be used when connecting to DB2k.

If you do not want to connect to DB2 with the servant region's userid, you must define an alias that holds a userid and password to be used on the getConnection( ) method. Click on Security > Configuration > J2C Authentication Data.

Click New and specify an alias, a userid and a password that can be passed to DB2 (or any other J2C resource), then click Apply.

Note: Add as many aliases as you require for access to DB2.

5. Define the DB2 for z/OS Local JDBC Provider (RRS) for data access.

   From the administrative console, click Resources > JDBC Providers. Select the server on which you want to install a JDBC provider and click Apply.

   Click New. In the list of JDBC providers, select **DB2 390 Local JDBC Provider (RRS)**, then click Apply.

6. Define a data source specifying the aliases for component and container connections. (optional)

   Select data source to create a data source for a J2EE 1.3 compliant application (creating a data source for a J2EE 1.2 compliant application is left as an exercise for the reader).

   Click New. Scroll down on the Data Sources panel, and specify for your data source:

   - A Name
   - A JNDI name
   - The database name, which is the location of the DB2 subsystem that manages the data you want to access
   - A Component-managed Authentication Alias (optional)
   - A Container-managed Authentication Alias (optional)

   Indicate if you want this data source to be used for container managed persistence, then click Apply. Save all of your changes to the master configuration.

   Now you are ready to install an application and test your data source.

Using a DB2 Universal JDBC Driver Provider with WebSphere Application Server for z/OS:

Verify that DB2 Universal JDBC Driver on z/OS and all the files required by WebSphere Application Server for z/OS are installed and available in an HFS directory before you continue with this configuration.

To use DB2 Universal JDBC Driver with WebSphere Application Server for z/OS, one of the following versions of the driver must be installed. Refer to the APARs and DB2 product information for details on installation.

- The DB2 Universal JDBC Driver in DB2 UDB for z/OS Version 8. This version supports both driver Types 2 and 4.
- The DB2 Universal JDBC Driver Provider by APAR PQ80841 on DB2 UDB for OS/390 and z/OS Version 7. This version supports both driver Types 2 and 4.
- The z/OS Application Connectivity to DB2 for z/OS feature that provides DB2 Universal JDBC Driver Type 4 connectivity. This version supports only driver Type 4 connectivity to DB2 databases. If this driver is installed, only the DB2 Universal JDBC Driver Provider (XA) can be used on WebSphere Application Server for z/OS to access remote DB2 databases.

Remember: Because DB2 constantly augments its support for WebSphere Application Server, consult DB2 service updates to see if the product offers an upgraded version of the driver that you need.

To use a DB2 Universal JDBC Driver with WebSphere Application Server for z/OS, you must:

- Configure the DB2 Universal JDBC Driver.
- Define a DB2 Universal JDBC Driver provider for WebSphere Application Server for z/OS.
- Define a DB2 Universal JDBC Driver provider data source.

1. Configure the DB2 Universal JDBC Driver for WebSphere Application Server for z/OS
   a. Define a DB2 Universal JDBC Driver provider

      Before you create a JDBC provider for the DB2 Universal JDBC Driver for z/OS, WebSphere Application Server for z/OS must know the location of the installed DB2 Universal JDBC Driver and license file, and the location of any native files that might be required by the DB2 Universal JDBC Driver. To do this, from the WebSphere Application Server for z/OS Administrative Console, go to Environment > Manage WebSphere Variables, and update the values of the following environment variables:

      1) **DB2UNIVERSAL_JDBC_DRIVER_PATH**

         Specify the fully-qualified path of the directory that contains the DB2 Universal JDBC Driver. This must be the directory that contains the db2jcc.jar and the db2jcc_license_cisuz.jar.

         Example: If the fully-qualified path of the db2jcc.jar is /usr/lpp/db2810/jcc/classes/db2jcc.jar, specify /usr/lpp/db2810/jcc/classes as the value of the variable.

      2) **DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH**

         Specify the fully-qualified directory path of the directory that contains the DB2 Universal JDBC Driver native files, if necessary. This is the directory that contains the driver files that have a .so file type. If the driver version you are using does not require native files, leave this value at null.

         Example: If the fully-qualified path of the directory containing the native files is /usr/lpp/db2810/jcc/lib, specify /usr/lpp/db2810/jcc/lib as the value of the variable.

   b. Bind the required DB2 packages

      As with any application that executes SQL statements in DB2 for z/OS, the Universal JDBC driver must first bind with DB2 the packages that represent the SQL statements to be executed. The Universal JDBC Driver does not use the same packages used by the legacy JDBC driver, and uses a different process for binding its packages.

      The specific details of the bind utility and bind process are described by the README provided with the installed DB2 Universal JDBC Driver. Refer to this README for details on how to setup and perform the required binding.

      Also note that the utility requires the server name (or IP address), the port number, and the database name (the database location on z/OS) for the target DB2. To get this information, issue a DB2 **-DISPLAY DDF** command on the target DB2 system. This displays the IPADDR (IP address), the SQL DOMAIN (server name), the TCPPORT number, and the LOCATION (database name/location) for you to use as input to the utility.

      You must perform the bind process for each target DB2 that will be accessed using the DB2 Universal JDBC Driver.

   c. Set up to handle in-doubt transactions

      You must perform this setup once for each target DB2 for z/OS Version 7 location that is accessed using the DB2 Universal JDBC Driver Type 4 XA support.

      Because DB2 for z/OS Version 7 does not implement J2EE XA support, the Type 4 driver XA processing uses DB2 V7 two-phase commit protocol and a table in each location (database) to store a list of global transactions that are in doubt (finished but not committed).

      This table must be set up at each DB2 V7 location that is accessed. To do this, use the In-Doubt Utility, which is included as part of the installed DB2 Universal JDBC Driver. Use this utility to create the SYSIBM.INDOUBT Table that stores information about In-Doubt Global Transactions. This utility also binds the package T4XAIndbtPkg, which contains the SQL statements to insert and delete from the SYSIBM.INDOUBT Table. The T4XAIndbtPkg package is written with SQLJ.

      This installation process requires that the target DB2 subsystem be configured with DDF enabled for incoming TCP/IP connections.

      1) To enable DDF on the target DB2, issue the DB2 **-START DDF** command on that system.
      2) This utility requires the server name (or IP address) and the port number for the target DB2 V7. To obtain this information, issue a DB2 **-DISPLAY DDF** command on the target DB2 V7

system. This displays the IPADDR (IP address), the SQL DOMAIN (server name), and the TCPPORT number that can be used as input to the utility.

To find more detailed information about the In-Doubt utility, refer to the *DB2 Universal Database for z/OS Version 7 Application Programming Guide and Reference for Java™* publication. (You can download it from the Library section of the DB2 Universal Database for z/OS Version 7 product information Web pages.) Within this publication, search for discussion about the utility under **DB2T4XAIndoubtUtil**, which is the official name of the In-Doubt utility.

Note: The previously described setup for in-doubt transactions is ***not*** a requirement for DB2 FOR z/OS Version 8 servers because DB2 FOR z/OS Version 8 natively supports XA commands over DRDA and manages the In-Doubt Global Transactions internally.

d. Define a db2.jcc.propertiesFile

A db2.jcc.propertiesFile for use by DB2 Universal JDBC Driver Type 2 processing under WebSphere Application Server for z/OS can be created and specified as input to the driver. This runtime properties file is for use in specifying various runtime options that the DB2 Universal JDBC Driver uses for Type 2 connectivity. These options are specified as properties in the form of parameter=value. Refer to the README file packaged with the installed DB2 Universal JDBC Driver for a detailed description of each of the properties.

This file is not required; however, if it is not provided, universal driver default processing is performed.

Of specific interest is the db2.jcc.ssid property. This property specifies the DB2 subsystem identifier (not location name), to be used by the DB2 Universal JDBC Driver Type 2 processing as the local subsystem name to which it should connect. If this property is not provided, the driver uses the subsystem identifier that it finds in the DSNHDECP load module. If the installation wants to use the DSNHDECP load module to specify the subsystem identifier, this load module must be included in a steplib dataset in the servant region PROCs associated with each server that will use the DB2 identified by the subsystem ID. Refer to the README file packaged with the universal driver for more information on using this load module. If that DSNHDECP load module does not accurately reflect the desired subsystem, or if multiple subsystems might be using a generic DSNHDECP, the db2.jcc.ssid property must be specified.

Although the db2.jcc.propertiesFile is not required, if you choose to define the file, you must specify the fully qualified-hfs-filename. To do this, specify the file as a JVM System property as follows:

- **db2.jcc.propertiesFile = <fully-qualified-hfs-filename>**

Because the driver-general properties are typically specific to a driver load (for example, server) versus to all servers using the JDBC provider, it is best that this JVM property be set at the server level. To define the **db2.jcc.propertiesFile=** property to the server level using the WebSphere Application Server for z/OS Administrative Console:

1) Under the WebSphere Application Server for z/OS Administrative Console, go to Servers > Application Servers, then click the server to which you want to add the JVM property.
2) From the selected server page, go to Process Definition > Servant.
3) On the Servant page, scroll down to the Additional Properties at the bottom of the page, then click Java Virtual Machine.
4) On the Java Virtual Machine page, scroll down to Additional Properties at the bottom of the page, then click Custom Properties.
5) On the Custom Properties page, scroll down to New to configure a new JVM property for the selected server. The name of the property is **db2.jcc.propertiesFile**. The value of the property is the fully-qualified-hfs-filename that you created and initialized with the DB2 Universal JDBC Driver properties. These are the properties that you want the Type 2 driver to use for the selected server
6) Click OK.
7) Click Save to save the new JVM property.

2. Define the DB2 Universal JDBC Driver provider.

After the DB2 Universal JDBC Driver is configured for WebSphere Application Server for z/OS, configure either a DB2 Universal JDBC Driver provider, which is non-XA, or a DB2 Universal JDBC Driver provider (XA), which supports XA. In doing so, it is important to note that if you use both the DB2 Legacy JDBC Driver and the DB2 Universal JDBC Driver under WebSphere Application Server for z/OS, you must ensure that DB2 JDBC providers associated with these two drivers are not located on the same server. (Refer to Provider Coexistence Considerations).

IBM suggests that you define your DB2 Universal JDBC Driver providers at a server level to reduce the chance of conflict with the DB2 for zOS Local JDBC Provider (RRS) that uses the DB2 Legacy JDBC Driver. Likewise, any DB2 for z/OS Local JDBC Provider you have defined must be defined at the server level to avoid conflict.

To define a DB2 Universal JDBC Driver provider:

a. From the WebSphere Application Server for z/OS Administrative Console, click Resources > JDBC Providers.
b. On the JDBC Provider page, set the JDBC Provider scope to the server upon which you want to install the new provider.
c. Click Apply.
d. Click New. The JDBC Providers page is displayed.
e. Make your selections from the three drop-down lists for either a non-XA or an XA implementation of the DB2 Univeral JDBC Driver provider:

   Non-XA implementation

   1) For database type, select DB2
   2) For provider type, select DB2 Universal JDBC Driver provider
   3) For implementation type, select Connection pool data source

   This provider supports driver Types 2 and 4. It does not support J2EE XA transaction processing. In the case of driverType 2 processing, RRS is used to coordinate transaction processing and manage global transaction using two-phase commit. In the case of driverType 4, one-phase commit processing is used to manage transactions.

   Note: Do not select this provider if your installation has the z/OS Application Connectivity to DB2 for z/OS feature defined to WebSphere Application Server for z/OS. This feature provides only universal driver type 4 XA connectivity, which is not supported by the non-XA DB2 Universal JDBC Driver provider.

   XA implementation

   1) For database type, select DB2
   2) For provider type, select DB2 Universal JDBC Driver provider
   3) For implementation type, select XA data source

   This provider supports only driverType 4. It uses J2EE XA to manage global transactions across multiple resource managers and to perform two-phase commit processing.

f. A configuration view of the selected provider displays, showing the default name of the provider, the classpath, the native library path, and the datasource implementation classname used by the provider. With the exception of the provider name, typically none of the information changes. If you choose, you can type your name for the provider in the Name field.
g. When the provider definition is complete, click Apply.
h. Finally, click Save to save the new JDBC provider.

3. Defining a DB2 Universal JDBC Driver provider data source

   To specify a data source for the defined DB2 Universal JDBC Driver provider:

   a. From the WebSphere Application Server for z/OS Administrative Console, click Resources > JDBC Providers. On the JDBC Providers page that displays, select the DB2 Universal JDBC Driver provider that requires the definition of a data source.

b. On the DB2 Universal JDBC Driver Provider page that displays, in the Additional Properties section at the bottom of the page, make a choice as follows:
   - Choose **Data Sources** if you want to define a data source for a DB2 Universal JDBC Driver provider (XA). In this case, Data Sources (Version 4) is not supported.
   - Choose **Data Sources** or **Data Sources (Version 4)** if you want to define a data source for a DB2 Universal JDBC Driver provider. This choice depends on the type of data source you want to define.

c. On the Data Sources page that displays, click New.

d. On the New page, supply values for these important properties (your security implementation, of course, does not require all of the alias types):
   - Name
   - JNDI name
   - Indicate if you want this data source to be used for container managed persistence.
   - Component-managed Authentication Alias (optional)
   - Container-managed Authentication Alias (optional)
   - Mapping-Configuration alias (optional)
   - Database name, which is the location name of the target database used when establishing connections with this data source
   - Driver type, which is the JDBC connectivity type used by the data source

     If you want to use a driverType 4, set the value to 4. If you want to use a driverType 2 , set the value to 2. If the data source is for the DB2 Universal JDBC Driver provider (XA), specify only driverType 4. Specification of driverType 2 in the case of the DB2 Universal JDBC Driver provider (XA) is not supported.
   - Server name, which is the TCP/IP address or host name for the Distributed Relational Database Architecture (DRDA) server.

     This property is required only if driverType is set to 4. This property is not used if driverType is set to 2.
   - Port number, which is the TCP/IP port number where the DRDA server resides.

     Provide a value for this property only if driverType is set to 4. This property is not used if driverType is set to 2.

   Note: If you set the driverType property for the data source to 4, an appropriate managed Authentication Alias must be specified. If you set the driverType property for the data source to 2 and no managed Authentication Alias is specified, the user identity currently associated with the thread at the time of a getConnection request is used as the identity associated with the connection.

e. Click Apply. A link to Custom Properties is now available on this page.

f. Click the Custom Properties link if you want to define additional settings for your data source.

g. After specifying the properties, click **Save** to save the new data source.

Creating and configuring a data source using the administrative console:

After you create a JDBC provider, you must create a data source to access the backend data store. Application components use the data source to access connection instances to your database; a connection pool is associated with each data source. The product supports two different versions of data source:

- Version 4.0, for use with the Enterprise JavaBeans (EJB) 1.0 specification and the Java Servlet 2.2 specification
- The latest standard version for use with more advanced releases of these specifications

1. Open the administrative console.

2. Click **Resources** > **JDBC Providers**.
3. Choose the JDBC resource provider under which you want to create your data source. The detail page for this provider is displayed.
4. Under Additional Properties, click the **Data Sources** link that is appropriate for your application. The Data sources or Data sources (Version 4) page is displayed.
5. Click **New** to display the Data source settings page.
6. Verify that all the required properties have valid values.

   For data sources of the latest standard version:

   a. Select a DataStoreHelper class name from the list entitled DataStoreHelpers provided by WebSphere Application Server, or leave the default selection as is. If you want to use a data store helper other than those available in the drop-down list, click **Specify a user-defined DataStoreHelper**. Type a fully qualified class name in the field that is provided.

   b. The next section of properties varies according to the database selection, provider type, and implementation that you chose for your JDBC provider. These properties are either required or highly recommended for your data source. Provide valid values for these settings if you do not want to accept the default values.

   c. Click **Component-managed Authentication Alias** if your database requires a user ID and password for a connection. This alias is used only when the application resource reference is using `res-auth = Application`.

      Important:(For components with `res-auth=Container`) Both the Container-managed Authentication Alias and Mapping-Configuration Alias settings are deprecated. They are superseded by the specification of a login configuration on the resource-reference mapping at deployment time. You must now use this login setting to define the aliases at deployment.

   d. If you chose XA Data Source as the implementation type of your JDBC provider, you need to specify the alias used during transaction recovery processing. An additional section entitled Authentication Alias for XA Recovery is available. Select either **Use Application Authentication Alias** to use the same value that you chose for component-managed authentication, or select **Specify:** to choose a different alias from the drop-down list.
7. Click **Apply** to view a page with your new data source settings. Additional properties and Related items sections are now available on this page. Additional properties contains the Connection pool, Custom properties, and WebSphere Application Server data source properties choices. (If you are using a Version 4 data source, however, you see only the first two choices.)

   a. Click on the first link to define settings that affect the behavior of the Java 2 Connector (J2C) connection pool manager.

   a. Go to the Custom properties page to view and modify additional properties that the database vendor might require for the connection of its product to an application server.

   b. Use the WebSphere Application Server data source properties page to input settings that exclusively affect the WebSphere Application Server connection to the database.

   c. The Related items section (applicable only to later version data sources, not Version 4 data sources) contains the J2C Authentication data entries choice. Here, you can specify a list of user IDs and passwords for J2C security to use.
8. Click **Save**.
9. Return to the data source page to confirm that your new data source is displayed in the list.

You are now ready to install the application for which you configured the data sources. During installation, you can bind resource references to these data sources.

Data source collection:

Use this page to create or modify a data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources**.

Note: If you are using the Enterprise JavaBean (EJB) 1.0 specification and the Java Servlet 2.2 specification, you must use the **Data sources (Version 4)** console page.

Name:

Specifies the display name of this data source.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for this data source.

Data type String

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 create a data source for association with your JDBC provider. Think of the data source as a pooled set of connections necessary for conducting transactions between your application and database.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources** > **New** (if you are creating a new data source) or > *data_source* (if you are viewing an established data source).

Note: If your application uses an Enterprise JavaBean (EJB) 1.1 or a Java Servlet 2.2 module, you must use the **Data sources (Version 4)** > *data_source* console page.

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)

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.

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

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*.

Choose from aliases defined under **Security**>**JAAS Configuration**> **J2C Authentication Data**.

To define a new alias not already appearing in the pick list:
- Click **Apply** to expose Related Items.
- Click **J2C Authentication Data Entries**.
- Define an alias.
- Click the connection factory name at the top of the *J2C Authentication Data Entries* page to return to the connection factory page.
- Select the alias.

Data type Drop-down list

Mapping-Configuration Alias (deprecated):

Allows users to select from the **Security** > **JAAS Configuration** > **Application Logins Configuration** list.

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*.

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

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

WebSphere Application Server data source properties collection:

Use this page to view the WebSphere Application Server data source properties. These properties apply to the WebSphere Application Server connection, rather than to the database connection.

To view this administrative console page, click **Resources** >**JDBC Providers** > *JDBC_provider* > **Data sources** > *data_source* > **WebSphere Application Server connection properties**.

Statement Cache Size:

Specifies the number of free statements that are cached per connection.

The WebSphere Application Server data source optimizes the processing of prepared statements. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times.

If the 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 prepared statements that are 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. For example, if the application has 5 SQL statements, set the statement cache size to 5, so that each connection has 5 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 Version 7.3, 9.2, or 9.3 without latest fix 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 WebSphere Connection Pooling: If checked, the application server sets up connect pools for this datasource.

Enable Database Reauthentication: If checked, there is not 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.

PreTest 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.

PreTest Connection Retry Interval: When **PreTest Connection** 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.

Data type	Integer

Data sources (Version 4):

Use this page to view the settings of a Version 4.0 style data source.

These Version 4.0 data sources use the WebSphere Application Server Version 4.0 Connection Manager architecture. All EJB 1.1 modules must use a Version 4.0 data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources (Version 4)**.

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

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 (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.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources (Version 4)** > *data_source*.

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

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

Custom properties:

Use this page to view and configure the custom properties of a J2EE resource provider.

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.

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” on page 599 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 view and set 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.)

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 Version 4.0 data source.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources (Version 4)** > *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 Version 4.0 data source.

To view this administrative console page, click **Resources** >**JDBC Providers**> *JDBC_provider* > **Data Sources (Version 4)** > *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 a data source for a clustered environment:

Use these steps to define a data source on multiple nodes that comprise a cluster.

The first part of this task defines a JDBC provider with a cluster scope setting. Be aware that all members of your cluster must run at least Version 6 of WebSphere Application Server to use this scope setting for the cluster. (See Administrative console scope settings for more information about scope settings in general.)

The cluster scope has precedence over the node and cell scopes. Create a data source for a cluster if you want the data source to:

- Be available for all the members of this cluster to use
- Override any resource factories that have the same JNDI name that is defined within the cell scope

1. Open the administrative console.
2. Click **Resources** > **JDBC Providers**. In the Scope section, note that the default scope setting is at the *node* level.
3. Click **Browse Clusters**. The JDBC providers > Select a Cluster Scope panel is displayed.
4. Select the cluster for which you want to define a data source, and click **OK**. The JDBC providers panel is displayed again.
5. Click **Apply**.
6. Click **New** to create a new JDBC provider at the cluster level. The class path for your new JDBC provider is already filled in; part of that class path is specified using a symbolic variable, for example: *${DB2390_JDBC_DRIVER_PATH}/classes/db2j2classes.zip*. Leave it at the default.
7. Finish creating the JDBC provider.
8. Click **Environment**.
9. Click **WebSphere Variables**.
10. For each node, select the symbolic variable used in the class path of your JDBC provider, and provide a value that is appropriate for the selected node. For example, if the class path of your JDBC provider uses the symbolic variable *${DB2390_JDBC_DRIVER_PATH}*, you might supply the value `/usr/lpp/db2` on one node and `/usr/lpp/db2710` on another node, depending on where your DB2 390 installation is located.
11. Click **DB2_JDBC_DRIVER_PATH** (this already exists by default). Here provide the path (in the *value* field) where *db2java.zip* exists on the selected node.
12. Click **Apply** and save the changes.

    Note: This variable must be defined on each node within the cluster.

You are now ready to configure your data source. For guidance, refer to “Creating and configuring a data source using the administrative console” on page 589.

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 561.

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:

   ```
   export CLASSPATH=$CLASSPATH:/WebSphere/V5R1M0/AppServer/lib/
    wsexception.jar:/WebSphere/V5R1M0/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 java.io.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.management.*;
import com.ibm.ws.exception.*;

/**
 * Creates a node-scoped resource.xml entry for a
 * DB2 for zOS Local JDBC Provider (RRS) DataSource
 * when WebSphere security is not enabled
 *
 * The datasource created is for CMP use.
 *
 * To run this example, the following must be done:
```

```
 *
 * 1)  Set the WAS_HOME environment variable to the location of
 *     your WebSphere Application Server for z/OS Configuration
 *     directory
 *
 *     Example:  export WAS_HOME=/WebSphereV5R1M0/AppServer
 *
 * 2)  Set the following environment variables:
 *
 *     export WAS_LIB=$WAS_HOME/lib
 *     export WAS_CLASSPATH=[DIRECTORY_CONTAINING_THIS_FILE]
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/jmxc.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wsexception.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/admin.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wasjmx.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_HOME/java/jre/lib/ext/mail.jar
 *     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/ibmjlog.jar
 *
 * 3)  Execute the following commands:
 *
 *     javac -classpath $WAS_CLASSPATH CreateDataSourceCMP.java
 *     java -classpath $WAS_CLASSPATH CreateDataSourceCMP
 */
public class CreateDataSourceCMP {

    String dsName           = "MyDataSourceCMP";   // ds display name , also jndi name and CF name
    String dbName           = "LOC1";           // database name
    String authDataAlias    = "IBMUSER";        // an authentication data alias
    String uid              = "IBMUSER";        // userid
    String pw               = "IBMUSER";        // password
    String dbclasspath      = "/db2beta/db2710/classes/db2j2classes.zip"; // path to the db driver
    String dblibpath        = "/db2beta/db2710/lib";              // path to the db lib directory

    /**
     * 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 = "My DB2 for zOS Local JDBC Provider (RRS) for CMP";
     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.jcc.DB2ConnectionPoolDataSource"));
     provAttrs.add(new Attribute("description","Legacy DB2 for z/OS driver using RRS"));

     //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);
     configService.addElement(session,jdbcProv,"nativepath",dblibpath,-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 CMP usage"));
     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 for zOS Local JDBC Provider (RRS) DataSource
```

```
* when WebSphere security is not enabled
*
* To run this example, the following must be done:
*
* 1)  Set the WAS_HOME environment variable to the location of
*     your WebSphere Application Server for z/OS Configuration
*     directory
*
*     Example:  export WAS_HOME=/WebSphereV5R1M0/AppServer
*
* 2)  Set the following environment variables:
*
*     export WAS_LIB=$WAS_HOME/lib
*     export WAS_CLASSPATH=[DIRECTORY_CONTAINING_THIS_FILE]
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/jmxc.jar
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wsexception.jar
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/admin.jar
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/wasjmx.jar
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_HOME/java/jre/lib/ext/mail.jar
*     export WAS_CLASSPATH=$WAS_CLASSPATH:$WAS_LIB/ibmjlog.jar
*
* 3)  Execute the following commands:
*
*     javac -classpath $WAS_CLASSPATH CreateDataSourceBMP.java
*     java -classpath $WAS_CLASSPATH CreateDataSourceBMP
*/
public class CreateDataSourceBMP {

   String dsName = "MyDataSourceBMP";  // ds display name , also jndi name and CF name
   String dbName = "LOC1";             // database name
   String authDataAlias = "IBMUSER";   // an authentication data alias
   String uid = "IBMUSER";             // userid
   String pw  = "IBMUSER";             // password
   String dbclasspath = "/db2beta/db2710/classes/db2j2classes.zip";  // path to the db driver
   String dblibpath = "/db2beta/db2710/lib";               // path to the db native library directory

   /**
    * 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();
       } 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 = "My DB2 for zOS Local JDBC Provider (RRS) for BMP";
      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.jcc.DB2ConnectionPoolDataSource"));
      provAttrs.add(new Attribute("description","Legacy DB2 for z/OS driver using RRS"));

      //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);
      configService.addElement(session,jdbcProv,"nativepath",dblibpath,-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 BMP usage"));
      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 ″J2EE Connector Architecture migration tips″ 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 561.

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
 }
}

puts "Finding the Resource Adapter"
set rsadapter [$AdminConfig list J2CResourceAdapter $serv]

#Now create a JDBC Provider for the data sources
puts "Creating the provider for COM.ibm.db2.jdbc.DB2XADataSource"
set attrs1 [subst {{classpath $driverClassPath} {implementationClassName COM.ibm.db2.jdbc.DB2XADataSource}
 {name "FVTProvider2"} {description "DB2 JDBC Provider"}}]
set provider1 [$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 SYSOUT of your application servant region for warning or exception messages.

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 the data sources that you configure for database access.

If the definition of your data source includes a WebSphere variable, you need to determine how your scope settings for both the variable and data source can affect the test connection results. Your next step is to choose which of the three ways you want to activate the test connection service: through the administrative console, the wsadmin tool, or a Java stand-alone program.

Verifying your scope settings

Use of a WebSphere variable in your data source definition can elicit test connection results that are incongruent with the actual run-time behavior of your data access application. In some cases, a test connection operation fails, but the data source functions normally during application run time. The reverse scenario can also occur. The cause of the potential conflict is the difference between how your application server handles WebSphere variable scope settings at run time, and how it handles those scope settings for a test connection operation. Understanding the difference helps you choose a successful configuration for your data source.

At run time, an application server resolves environment variables from the most specific scope to the broadest scope. That is, the server checks for the resolution of a variable in the server scope, then the cluster scope, then the node scope, and lastly the cell scope. When testing a connection, however, the most specific scope level at which the server checks variable definitions is determined by the scope at which the data source was created.

The test connection operation itself is performed in the JVM that corresponds with the scope of the data source. If the data source is configured at the cell scope, the test connection operation is carried out in the deployment manager process. If the data source is configured in the node scope, the test connection operation is performed in the node agent process for that node. For cluster-scoped data sources, the test connection operation is attempted in the node agent for each node that contains a cluster member. For server-scoped data sources, the test connection operation is first attempted in the server. If the server is unavailable, the test connection operation is retried in the node agent for the node that contains the application server.

Note: For any data source, regardless of scope setting, you must restart the associated JVM if you add or modify the authentication alias object.

Processing data source configurations in this way yields different, sometimes misleading, test connection results for different scope settings, as shown in the following table:

Table 5. Test connection results for different data source and environment variable combinations

Data source scope	Cell level variables	Node level variables	Server level variables
Cell level	Ok* *(See the following section)*	Fail	Fail
Node level	Ok	Ok	Fail
Server level	Ok	Ok	Ok

Contrary to expectations, these test connection failures generally do not translate into run-time failures, assuming that you are conscientious about configuring your WebSphere variables. A `variable cannot be found` exception results from attempted use of a data source that is configured with undefined variables.

Test connection success, but run-time failure

One of the scope combinations listed in the table, however, produces the reverse scenario: the test connection operation succeeds, but the data source fails at run time. This anomaly occurs in the case of a cell-scoped data source that uses a cell-scoped WebSphere variable. At run time, the cell-scoped variable is preempted by a default scope setting. In a default WebSphere Application Server installation, the supported database driver variables are defined and initialized to an empty string at the node scope. That empty setting effectively overrides any variable definition you provide at the cell scope level for a cell-scoped data source.

Because the run-time server checks the node scope for the variable before it checks the cell scope, it reads the empty string variable and accepts it as a value for the database driver class path. The incorrect

class path elicits a `classNotFound` exception when the server attempts to use the data source. To make the data source work for both test connection and run time, define the driver class path variable at the cell scope (indicating the location of the driver files on the deployment manager node), *as well as* at each node on which the data source must function. Alternatively, you can delete the node scope definitions if the database location is the same as the cell-scoped variable.

Bypassing variable lookups

You can bypass the environment variable lookups by changing the class path entries for the JDBC provider to hard-coded values. However, this strategy succeeds only if the class path is the same on all nodes where the data source must function.

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 Network Deployment installation of Application Server, you connect to the JMX server running in the deployment manager, usually running on port 8879.

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. On the collection panel, you can select multiple data sources and test them all at once. Be sure that the following conditions are met before using the Test Connection button.

1. Depending on your specific needs, a valid *Authentication Data alias* might need to exist and be supplied on the data source panels.
2. 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 filled in.
3. If you used a WebSphere environment entry for the classpath or other fields, such as *${DB2390_JDBC_DRIVER_PATH}/classes/db2j2classes.zip*, make sure that you assign it a value in the *WebSphere Variables* page. Note that if you add a new WebSphere environment variable, or modify it , the process (node agents and deployment manager) might need restarting.
4. Verify that the environment variables used exist in the scope of the test. For example, if the node scoped data source you defined uses *${DB2_JDBC_DRIVER_PATH}*, check that there exists a node level definition for *DB2_JDBC_DRIVER_PATH = c:\sqllib\java*(or your system dependent value).
5. Ensure that the deployment manager and node agent are up and running.
6. 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.
   a. 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.
   b. 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.
   c. 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. It takes a *configuration ID* as an argument.

Note: There is no way to pass a user ID and password to this invocation. 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 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**.

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 Providers** > *JDBC_provider* > **Data Sources** > *data_source* > **Connection Pool**.

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 `ConnectionWaitTimeout` exception is issued.

For example, if the Maximum 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 Maximum Connections is set to 0, the connection pool is allowed to grow infinitely. This also has the side effect of causing the Connection Timeout value to be ignored.

If multiple standalone application servers use the same data source, there is one pool for each application server. 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

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. 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**. JCA data sources can have either option. WebSphere Version 4.0 data sources always have a purge policy of **EntirePool**.

Data type String

Default EntirePool

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.

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.

Partition support is always enabled. The default values of 0 should be used enabling 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.

To view this administrative console page, click **Resources** > **Resource Adapters** > *resource_adapter* > **J2C Connection Factories** > *connection_factory* > **Connection Pool** > **Advanced Connection Pool**.

Number of shared partitions:

Specifies the number of partitions that are created in each of the shared pools.

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:

The free pool distribution table size is used for better distribution of the Subject and CRI hash values within a hash table to minimize collisions for faster retrieval of a matching free connection.

If there are many incoming requests with varying credentials, this value can help with the distribution of finding a free pool for a connection for that user. Larger values are more common for installations that have many different credentials accessing the resource. Smaller values (1) should be used if the same credentials apply to all incoming requests for the resource.

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.

To view this administrative console page, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data Sources (Version 4)** > *data_source* > **Connection Pool**.

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

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 issuing a `ConnectionWaitTimeout` exception to the application.

Setting this value to 0 disables the connection timeout.

Data type Integer
Units Seconds
Default 180
Range Any non-negative integer

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:

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 select a connection factory, which represents one set 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.

To view this administrative console page, click **Resources** > **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

Description:

Specifies a text description of this connection factory.

Data type String

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.

To view this administrative console page, click **Resources** > **Resource Adapters** > *resource_adapter* > **J2C Connection Factories** > *connection_factory*.

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

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

Component-managed Authentication Alias:

Specifies authentication data for component-managed signon to the resource.

Choose from aliases defined under **Security**>**JAAS Configuration**> **J2C Authentication Data**.

To define a new alias not already appearing in the pick list:
- Click **Apply** to expose Related Items.
- Click **J2C Authentication Data Entries**.
- Define an alias.
- Click the connection factory name at the top of the *J2C Authentication Data Entries* page to return to the connection factory page.

- Select the alias.

Data type Pick-list

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*.

Choose from aliases defined under **Security**>**JAAS Configuration**> **J2C Authentication Data**.

To define a new alias not already appearing in the pick list:
- Click **Apply** to expose Related Items.
- Click **J2C Authentication Data Entries**.
- Define an alias.
- Click the connection factory name at the top of the *J2C Authentication Data Entries* page to return to the connection factory page.
- Select the alias.

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):

Allows users to select from the **Security** > **JAAS Configuration** > **Application Logins Configuration** list.

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*.

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

J2C Connection Factory advanced settings:

Use this page to specify settings for a connection factory.

To view this administrative console page, click **Resources** > **Resource Adapters** > *resource_adapter* > **J2C Connection Factories** > *connection_factory* > **Advanced Connection Factory Properties**.

Manage cached handles:

Specifies whether cached handles (handles held in `inst vars` in a bean) should be tracked by the container.

Data type Checkbox

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

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.

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.

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 Web 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*.

Configuring Cloudscape Version 5.1.60x

Cloudscape provides the following two separate frameworks for running Cloudscape with WebSphere Application Server:

- Embedded: This framework is the same as the one for Cloudscape Version 5.0. To use this framework, define a Cloudscape JDBC provider. See the Cloudscape section in the minimum required settings article for more information.

  You must use the embedded framework if you are running XA. Cloudscape does not support XA on Network Server.
- Network Server: This framework was a new feature in Cloudscape Version 5.1, and removes these limitations that existed in earlier versions of Cloudscape:
  - inability to access a remote Cloudscape instance
  - only one JVM can boot up the same database instance

  The following steps describe how to configure and run the Network Server framework.

1. Start the Network Server on the machine that hosts the database instance.

   To start the Network Server, run the **startNetworkServer.bat** file, which is located in the `WAS_HOME/cloudscape/bin/networkserver` directory. On UNIX platforms, the file is **startNetwokServer.sh**.
2. Update the **db2j.properties** file, which is located in the `WAS_HOME/cloudscape` directory, if necessary.

   Cloudscape should work without any modifications to this file.

   Use the entries in the **db2j.properties** file to turn on trace, change the port number on which Network Server listens, and enable other functions of the Network Server framework. The default port number on which the Network Server listens is port 1527.

   For more information on this file, see the Cloudscape documentation at www.ibm.com/software/data/cloudscape/pubs/collateral.html.
3. Define a Cloudscape Network Server using Universal JDBC driver to connect Cloudscape with WebSphere Application Server using the Network Server framework.
4. Stop the Network Server by invoking the **stopNetworkServer.bat** file.

   You can find this file in the `WAS_HOME/cloudscape/bin/networkserver` directory. On UNIX platforms, the file is **stopNetworkServer.sh**.
5. Review additional tools available in the Network Server framework.

   Find these tools in the `WAS_HOME/cloudscape/bin/networkserver` directory. These tools are:
   - `sysinfo`
   - `cview`
   - `ij`
   - `dropSYSIBM` Use this tool to drop the SYSIBM schema and its contents.
6. Create a SYSIBM schema.

   If you do not create the SYSIBM schema, you cannot see the datatypes when you create tables using the `cview` graphical user interface. The **db2j.drda.loadSYSIBM** property in the **db2j.properties** file controls whether the schema is created on the first connection to the database. The **db2j.drda.loadSYSIBM** property default value is true.

   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.

Cloudscape Version 5.1.60x post installation instructions:

After installing Cloudscape Version 5.1.60x, you must complete the following steps before you can access the database.

If you are running a WebSphere Application Server Network Deployment configuration, you must ensure the correct server or scope is set before completing these steps.

1. Upgrade or migrate any existing database instances.
   a. Backup an existing database.

      You must complete a backup in case you have to access the previous version of Cloudscape. After you migrate a database, you cannot access your old database unless you perform a backup.
   b. Migrate an existing database by doing the following:
      - Set the ***connectionAttributes*** custom property to **upgrade=true**.

        The data source is located in the WebSphere Application Server administrative console under the JDBC providers.
      - If you are using the cview interface, located in the WAS_HOME/cloudscape51/bin/embedded directory, click **yes** when you see the *upgrade database* prompt.

        Note: Ensure you migrate **defaultDB**, which is located in the WAS_HOME/bin/DefaultDB directory.
2. Set or change the class path definitions in any existing JDBC providers, which are defined to use Cloudscape. Cloudscape JAR files will not load when WebSphere Application Server is active.

   Use the WebSphere Application Server environment variable **${CLOUDSCAPE_JDBC_DRIVER_PATH}\db2j.jar** to point to the new version of Cloudscape.

   The **CLOUDSCAPE_JDBC_DRIVER_PATH** environment variable is defined in WebSphere Application Server with a value of WAS_HOME/cloudscape/lib.

   In a Network Deployment configuration, you must ensure the correct server or scope is set for this variable to take effect. Typically the scope is the server on which you are running Cloudscape.
3. If the application server is running Cloudscape as a persistent store for UDDI in previous versions, additional steps are necessary.

   The server SystemOut log might issue this message:

   ```
   The data source class name com.ibm.db2j.jdbc.db2jConnectionPoolDataSource could not be found.
   ```

   This is because the Cloudscape JAR file has moved to from its location in version 5.x to a new location in version 5.1x.

   To correct this situation, do the following:
   a. Upgrade the database to Cloudscape 5.1.60x.
   b. Rerun the install script, ***or*** edit the class path field in the data source.

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.

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.

Connector modules collection

Use this page to view established connector modules, which are resource adaptor (RAR) files that have been packaged into deployable components compliant with the J2EE Connector Architecture (JCA).

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

You must generate a connector module for every resource adapter (RAR file) in the application. You do this through an assembly tool, which creates an instance of the connector module object for the RAR file. To learn more about the process, see the topic ″Assembling resource adapter (connector) modules″ in the Information Center.

Remove: Removes a module from the deployed application. The module is deleted from the application in the WebSphere Application Server configuration repository and also from all 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 then the application is restarted.

Update: Opens a wizard that helps you update a 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 same 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 a new module file is *added* (that is, established for the first time) 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 then the application is restarted.

URI:

Specifies the logical path to the resource that will be serviced by the product.

Name:

Specifies the display name of the connector module.

Connector module settings:

Use this page to view the settings of connector modules, which are resource adapter (RAR) files that have been packaged into deployable components compliant with the J2EE Connector Architecture (JCA).

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application* > **Connector Modules** > *connector_module*.

The following field descriptions refer to properties that are set when you create connector modules using an assembly tool. To learn more about the process, see the topic ″Assembling resource adapter (connector) modules″ in the Information Center.

Remove: Removes a module from the deployed application. The module is deleted from the application in the WebSphere Application Server configuration repository and also from all 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 then 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 then the application is restarted.

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 then the application is restarted.

Uri:

Specifies the logical path to the resource that is serviced by WebSphere Application Server.

Data type String

Name:

Specifies the display name of the connector module.

Data type String

altDD:

Specifies the alternate DD of the connector module.

The alternate DD URI for a given module.

Data type String

Starting weight:

Specifies the startup priority of the connector module over others.

When your application contains multiple modules, the starting weight you specify determines this module's startup priority over other modules during server startup. Modules with lower startup order are started first.

Data type String

Messaging resources

Using asynchronous messaging

These topics describe how to use asynchronous messaging with WebSphere Application Server, to enable enterprise applications to use JMS resources and message-driven beans.

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. This JMS support 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 670
- "Using JMS resources of WebSphere MQ" on page 704
- "Using JMS resources of a generic provider" on page 770
- "Administering support for message-driven beans" on page 779
- ″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.

Besides using the programming interfaces directly to explicitly poll for messages, WebSphere Application Server also supports the use of message-driven beans as asynchronous message consumers. A

message-driven bean is invoked by the EJB container when a message arrives at the destination that it is configured to listen on, without an application having to explicitly poll the destination.

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 for that purpose. In the JCA 1.5 specification, such message-driven beans are commonly called message endpoints or simply endpoints.

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 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 a 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).

You can use the WebSphere administrative console to administer the WebSphere Application Server support for asynchronous messaging. For example, you can configure JCA resource adapters, J2C activation specifications, JMS providers, and JMS resources, and can control the activity of messaging services.

To learn more about WebSphere messaging support, see the following topics:

JMS providers

This topic provides an overview of the support for JMS providers by 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 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.

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).

Using JMS interfaces to explicitly poll 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.

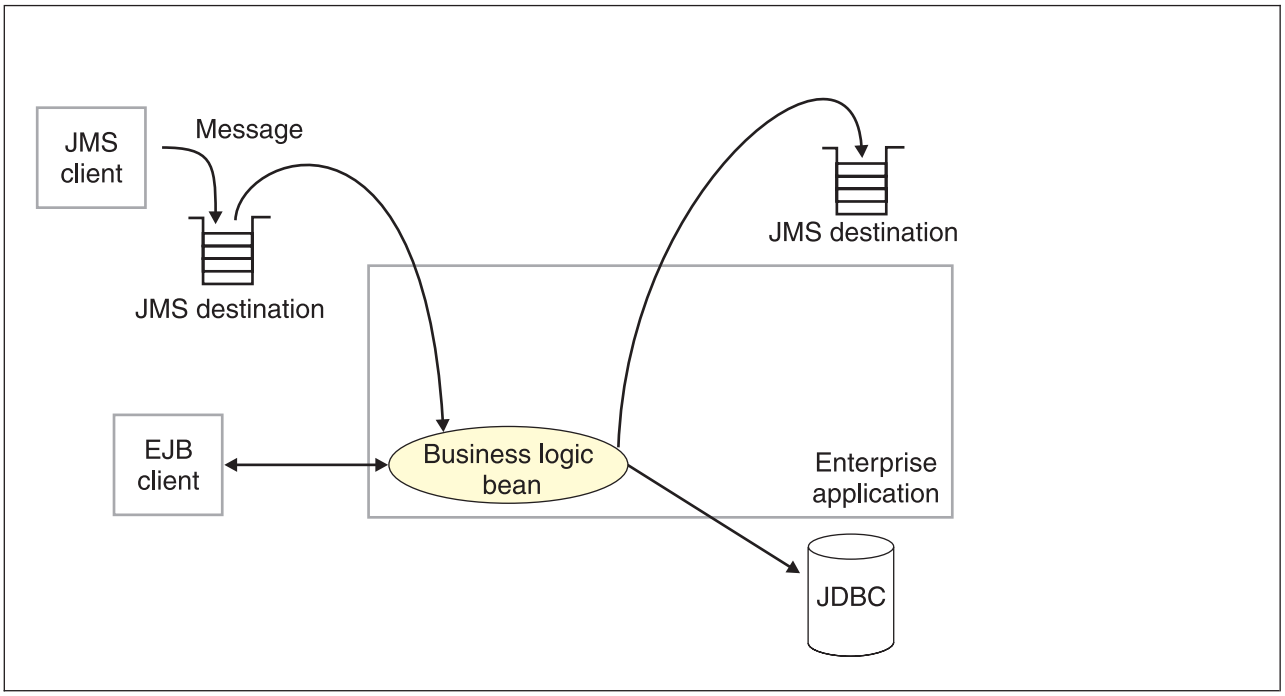


Figure 3. Asynchronous messaging using JMS. This figure shows an enterprise application polling a JMS destination to retrieve an incoming message, which it processes with a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to 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.

Using message-driven beans to automatically retrieve messages

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 ″Enterprise bean component security″ in the information center.

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).

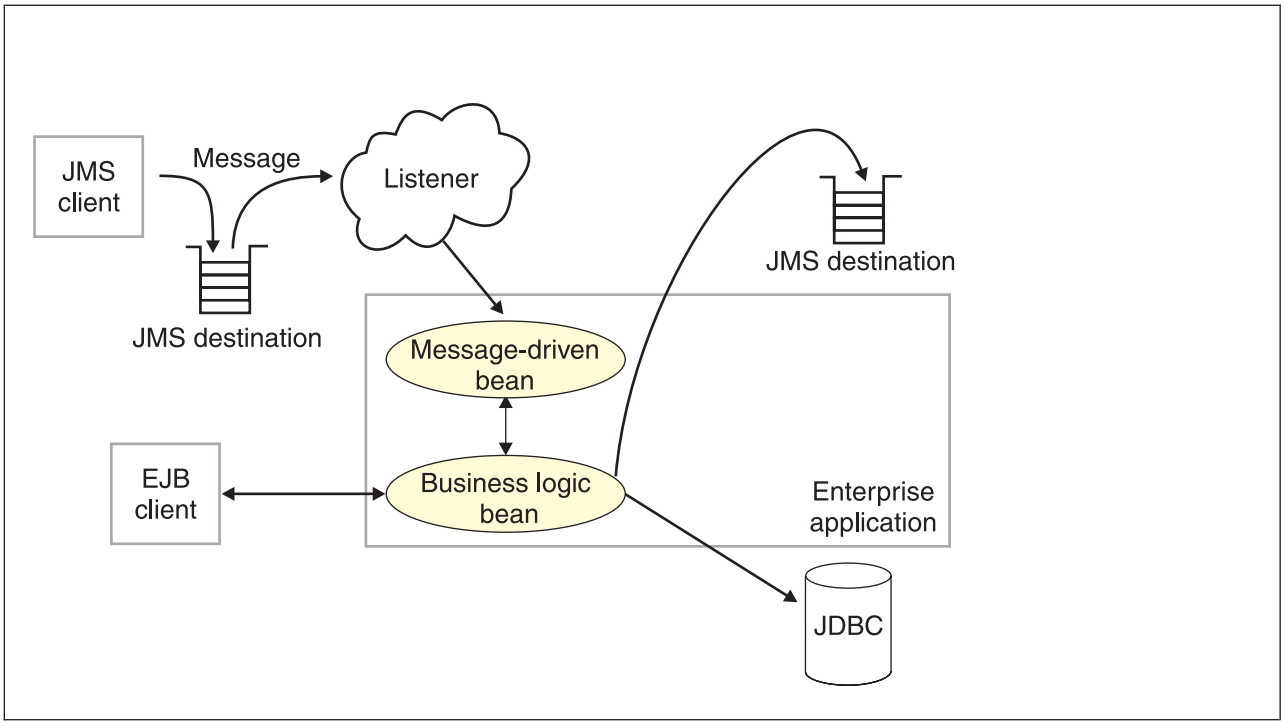


Figure 4. Messaging with message-driven beans. This figure shows an incoming message being passed automatically to the onMessage() method of a message-driven bean that is deployed as a listener for the destination. The message-driven bean processes the message, in this case passing the message on to 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.

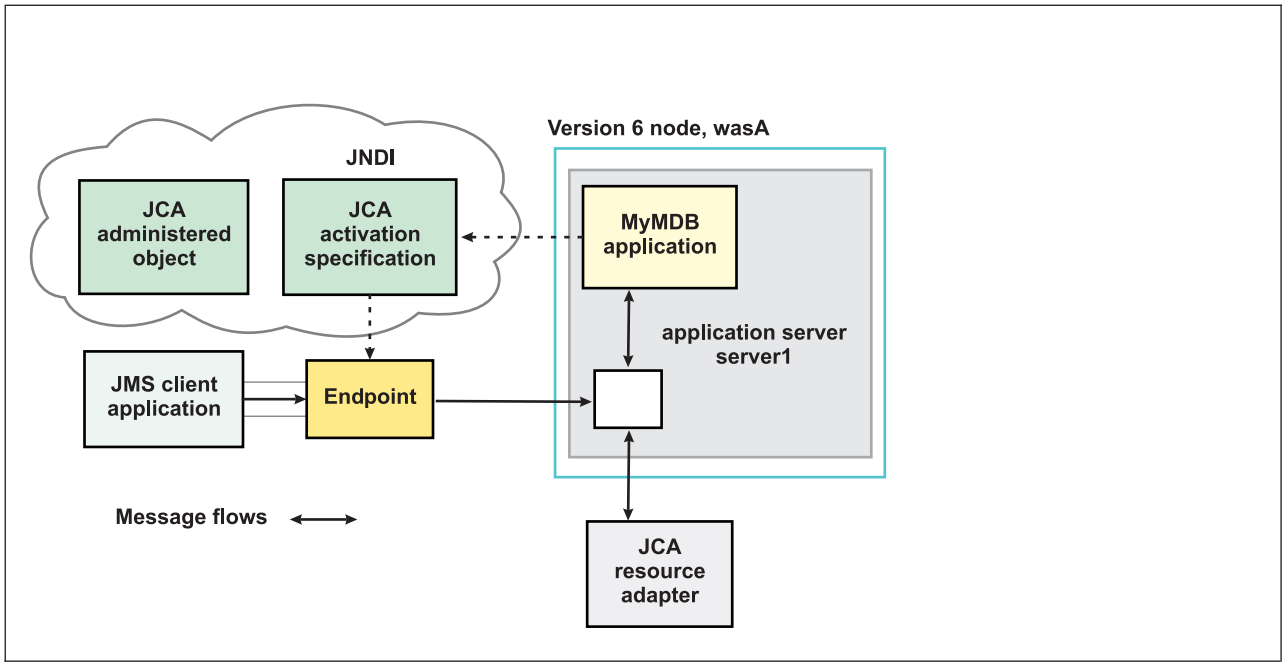


Figure 5. Message-driven bean components for a JCA resource adapter. This figure shows the main components of WebSphere support for message-driven beans for use with an external JCA resource adapter.

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.

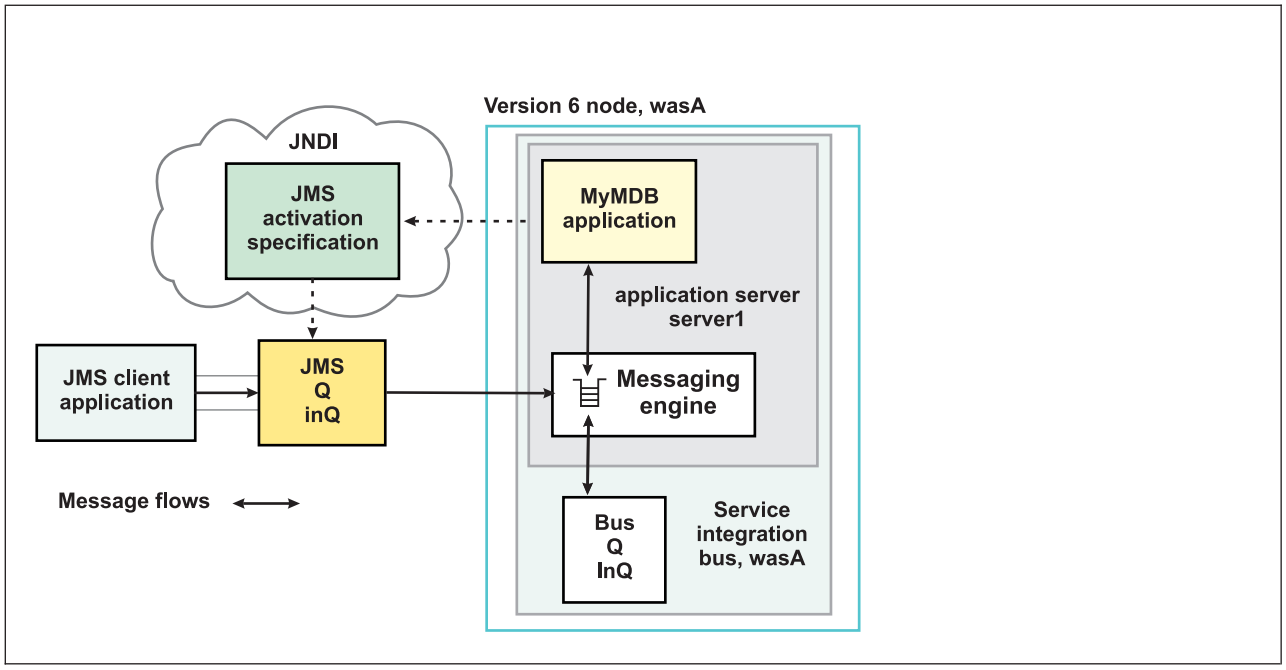


Figure 6. Message-driven bean components for the default messaging provider. This figure shows the main components of WebSphere support for message-driven beans for use with the default messaging provider.

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″ in the information center.

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 `messagelistener` types in its deployment descriptor (ra.xml). The messagelistener type 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 messagelistener-type interfaces provided by the resource adapter. By allowing multiple message listener types, 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 messagelistener-type 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 messagelistener type 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).

J2C activation specification configuration options and precedence:
Resource adapter scoped configuration

A J2C activation specification configuration instance can be created and modified under an installed resource adapter at the cell, node, or server scope. This activation specification configuration is created based on a particular message listener type for the given resource adapter. Valid properties available for configuration are determined by introspection of the ActivationSpec class instance provided with the resource adapter. When created, an ActivationSpec class instance is referenced by its JNDI name. This activation specification configuration is needed during the deployment of a message-driven bean for the resource adapter.

Configuring a J2C activation specification instance at the cell, node, or server level offers two distinct advantages:

- The activation specification configuration information can be share among multiple message-driven beans across multiple applications.
- Updates to the configuration properties can be made without the need to redeploy the application.

Application-based configuration

Applications with message-driven beans have the option of specifying all, some, or none of the properties needed by the ActivationSpec class. 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.

To deploy and activate a message-driven bean with respect to application specification configuration properties would be as follows:

1. Use JNDI to look up a J2C activation specification configuration instance, which is based on its resource adapter-scoped definition.
2. Set the properties needed by the ActivationSpec class to the values defined by the cell, node, or server definition. If any of the properties are also defined as activation-config properties of the application, use the value defined by the activation-config property.
3. During application startup, the server activates the MDB endpoint by calling the resource adapter and passing a configured instance of the ActivationSpec.
4. Note that a resource adapter can specify in its deployment descriptor if a given ActivationSpec property is required. If it is required, and it is not supplied either by the cell, node or server definition, or an activation-config property from the applications deployment descriptor, then an exception is raised as part of the sequence to activate the message-driven bean.

WebSphere activation specification optional binding properties:

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.

WebSphere Application Server cloning and WebSphere MQ clustering

This topic provides a summary of information about using WebSphere Application Server horizontal cloning with WebSphere MQ server clustering support. It describes a scenario that shows how the message listener service can be configured to take advantage of WebSphere MQ server clustering and provides some information about how to resolve potential runtime failures in the clustering scenario. The information in this topic is based on the scenario shown in the figure WebSphere Application Server horizontal cloning with WebSphere MQ clustered queues.

Note: WebSphere MQ server clustering is only available with the full WebSphere MQ product installed as a JMS provider.

Each JMS listener is used to retrieve messages from a destination defined to the server. In the following information the listener configurations are the same for each WebSphere application server. Each application server host contains a WebSphere application server and an WebSphere MQ server. If a host is only used to distribute messages, it only contains an WebSphere MQ server. There can be many servers defined in the configuration, although for simplicity the information in this topic is based on a scenario containing only three servers as shown in "WebSphere Application Server cloning and WebSphere MQ clustering."

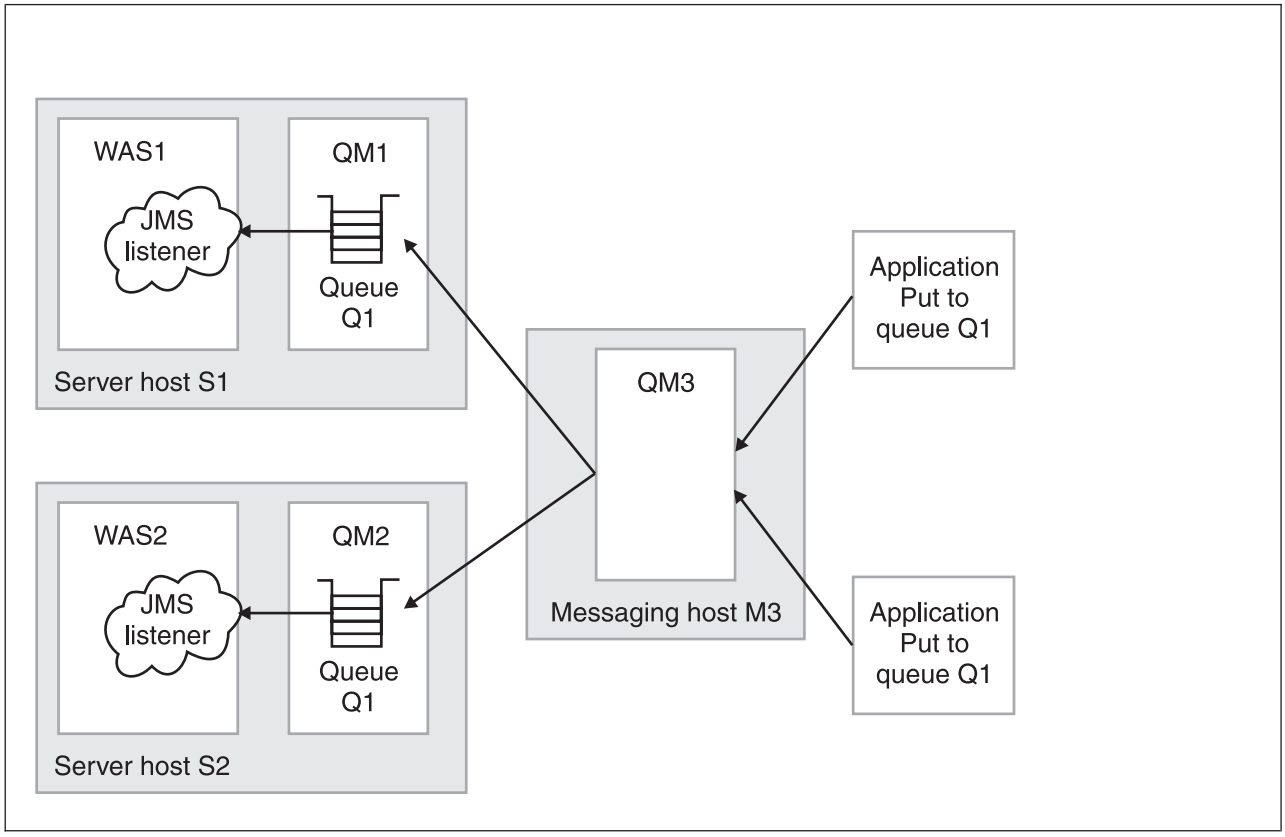


Figure 7. WebSphere Application Server horizontal cloning with WebSphere MQ clustered queues. This figure shows two WebSphere Application Server hosts, with horizontal clustering, and a messaging host used to distribute messages for WebSphere MQ server clustering. For more information, see the text that accompanies this figure.

The scenario shown in "WebSphere Application Server cloning and WebSphere MQ clustering" comprises the following three hosts:

- Server host S1 contains the following servers:
  WebSphere MQ server.
  The server is defined to have a queue manager, QM1, and a local queue, Q1. The queue

manager belongs to a cluster. The queue is populated by the WebSphere MQ server located on host M3. Applications can add messages directly to the queue, Q1, but would not be subjected to the control of the WebSphere MQ cluster.

WebSphere Application Server

This contains a cloned application server, WAS1, which is configured with a JMS listener. The listener is configured to retrieve messages from JMS destination Q1.

- Server host S2 contains the following servers:

  WebSphere MQ server.

  The server is defined to have a queue manager, QM2, and a local queue, Q1. The queue manager belongs to the same cluster as QM1 on host S1. As with QM1, the queue is populated by the WebSphere MQ server located on host M3. Applications can add messages directly to the queue, Q1, but would not be subjected to the control of the MQ cluster.

  WebSphere Application Server

  This contains a cloned application server, WAS2 (identical to WAS1 on host S1), which is configured with a JMS listener. The listener is configured to retrieve messages from JMS destination Q1.

- Messaging host M3 contains the following servers:

  WebSphere MQ server.

  The server is defined to have a queue manager, QM3, which also belongs to the same cluster as QM1 and QM2. Applications add messages to the queue manager and queue Q1. The cluster to which this queue manager belongs causes messages to be distributed to all other queue managers in the cluster which have queue Q1 defined.

  Note: Queue Q1 is not defined as a local queue on this host. If the queue was defined locally, then messages would remain on the server for local processing; messages would not be distributed by the queue manager cluster control to the other queue managers in the cluster that do have the queue defined.

  This host does not have a WebSphere application server defined. All message retrieval processing is performed by the other two application servers on hosts S1 and S2.

Recovery scenarios

There are several failure scenarios that could occur with the clustering configuration; for example:

- WAS server failures.

  In this scenario the failure of any single WebSphere application server results in the messages for the specified destination remaining on the queue, until the server is restarted.

- WebSphere MQ Queue Manager failures.

  There are two different failures to consider:

  1. Failure of a queue manager on the same host as a WebSphere application server (for example, failure of QM2 on host S2). In this case messages are delivered to the other available application servers, until the WebSphere MQ server is back online, when messages are processed as expected.
  2. Failure of the messaging host M3 and its queue manager, QM3. In this case, the result of an outage is more significant because no messages are delivered to the other queue managers in the cluster. In a fully-deployed and scaled production system, host M3 would not be designed to be a single point of failure, and additional messaging servers would be added to the cluster configuration.

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 operates as a part of the WebSphere Application Server global security, and is enabled only when global security is enabled.

When global 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″ in the information center.

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 644.

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 define JMS resources for the default messaging provider.

For more information about the default messaging provider, see Using the default messaging provider.

Configuring messaging with scripting

This topic contains the following tasks:

- "Configuring the message listener service using scripting" on page 657
- "Configuring new JMS providers using scripting" on page 658

Configuring the message listener service using 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 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

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

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

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

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

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

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

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 queue connection factories using 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 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

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

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 create MQQueue $newjmsp $mqqAttrs $template
     ```

   - Using Jython:

     ```
     print AdminConfig.create('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

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.

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.

In a deployment manager cell, you can use V5 Default Messaging resources to maintain Version 5 default messaging resources for both version 6 and version 5 nodes in the cell.

- V5 Default Messaging configured against a version 6 node provides a JMS transport to a messaging engine of a service integration bus that supports the version 6 default messaging provider. The messaging engine emulates the service of a version 5 JMS server.
- V5 Default Messaging configured against a version 5 node provides a JMS transport to 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 693
- "Managing Version 5 JMS servers in a deployment manager cell" on page 696
- "Configuring authorization security for a Version 5 default messaging provider" on page 697
- "Tuning JMS destinations" on page 701

Listing version 5 default messaging resources

Complete this task to display administrative lists of JMS resources for use by WebSphere application server version 5 applications.

You can use the WebSphere administrative console to display lists of the following types of JMS resources for use by WebSphere application server version 5 applications. You can use the panels displayed to select JMS resources to configure, administer, create, or delete (where appropriate).

To display administrative lists of Version 5 default JMS resources, complete the following general steps:

1. Start the WebSphere administrative console.
2. Display the version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS Providers** → **V5 Default Messaging**.
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. 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.
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 settings:

Use this panel to view the configuration properties of the selected JMS provider. *You cannot change these properties.*

To view this page, use the administrative console to complete one of the following steps:

- In the navigation pane, click **Resources** → **JMS Providers** → **WebSphere MQ**.
- In the navigation pane, click **Resources** → **JMS Providers** → **V5 Default Messaging**.
- In the navigation pane, click **Resources** → **JMS Providers** → **Generic** → *provider_name*.

If you want to browse or change JMS resources of the JMS provider, complete the following steps:

1. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications.
2. Under Additional Properties, click the link for the type of resource . For more information about the administrative console panels for the types of JMS resources, see the related topics.

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.

When JMS resources are created for this messaging provider, they are always created into the provider scope selected in this panel. To browse or change resources in other scopes, select the required level option, then click **Apply**, before clicking the link for the type of resource.

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.

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.

Data type String

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String
Default WebSphereJMSProvider

Description:

A description of the JMS provider, for administrative purposes within IBM WebSphere Application Server.

Data type String

Classpath:

The Java classpath for WebSphere MQ as a JMS provider. The list of paths or JAR file names that together form the location for the JMS provider classes.

[JMS Providers > WebSphere MQ and JMS Providers > Generic only]

Data type String
Default **[WebSphere MQ]** $MQJMS_LIB_ROOT

Native Library Path:

The native library path for WebSphere MQ as a JMS provider. An optional path to any native libraries needed by the JMS provider.

[JMS Providers > WebSphere MQ and JMS Providers > Generic only]

Data type String
Default **[WebSphere MQ]** $MQJMS_LIB_ROOT

The Native Library Path property is set to the directory where the WebSphere MQ Java feature is installed.

External initial context factory:

The Java classname of the initial context factory for the JMS provider.

[JMS Providers > Generic only]

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.

[JMS Providers > Generic only]

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.

Before you set any of the values on this panel, configure the JMS Server for each node by using the WebSphere Application Server ISPF Customization Dialog.

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

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 global 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 global 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 not 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.

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 global 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 global 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 not 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.

In a deployment manager cell, you can use V5 Default Messaging resources to maintain Version 5 default messaging resources for both version 6 and version 5 nodes in the cell.

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 Providers** → **V5 Default Messaging**.
2. **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.
3. In the content pane, under Additional Properties, click **WebSphere queue connection factories** This displays any existing JMS queue connection factories for the Version 5 messaging provider in the content pane.

4. 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.
5. **Optional:** Change properties for the queue connection factory, 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.

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 Providers** → **V5 Default Messaging**.
2. **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.
3. In the content pane, under Additional Properties, click **WebSphere topic connection factories** This displays any existing JMS topic connection factories for the Version 5 messaging provider in the content pane.
4. 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.
5. **Optional:** Change properties for the topic connection factory, 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.

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 optimize performance, configure the topic destination properties to best fit your applications. For more information, see “Performance considerations for WebSphere Version 5 queue destinations” on page 702.

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 Providers** → **V5 Default Messaging**.
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. In the content pane, under Additional Properties, click **WebSphere queue destinations** This displays any existing queue destinations for the Version 5 default messaging provider in the content pane.
4. 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.
5. **Optional:** Change properties for the queue destination, according to your needs.
6. Click **OK**.
7. Save any changes to the master configuration.
8. 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.
9. 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 optimize performance, configure the topic destination properties to best fit your applications. For more information, see “Performance considerations for WebSphere Version 5 topic destinations” on page 702.

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 Providers** → **V5 Default Messaging**.
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. In the content pane, under Additional Properties, click **WebSphere topic destinations** This displays any existing JMS topic destinations for the Version 5 default messaging provider in the content pane.
4. 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.
5. **Optional:** Change properties for the topic 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.

Managing Version 5 JMS servers in a deployment manager cell

Use this task to manage JMS servers on WebSphere Application Server version 5 nodes in a deployment manager cell.

The use of JMS servers is only intended to support migration from WebSphere Application Server version 5 nodes to WebSphere Application Server version 6.

In a WebSphere Application Server deployment manager cell, each Version 5 node can have at most one JMS server, and any Version 5 application server within the cell can use JMS resources served by any of those JMS servers. Applications on WebSphere Application Server version 6 can also use JMS resources served by any of those JMS servers.

You can use the WebSphere administrative console to display a list of all version 5 JMS servers, to show and control their runtime status. You can also 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: In general, the default values of properties for the version 5 default messaging provider are adequate for JMS servers. However, if you are running high messaging loads, you may need to change some WebSphere MQ properties; for example, WebSphere MQ 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.

To manage a version 5 JMS server, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers** → **JMS Servers** This displays a table of the JMS servers, showing their runtime status.
2. **Optional:** If you want to change the runtime status of a JMS server, complete the following steps:
   a. In the table of JMS servers, select the JMS servers that you want to act on.
      - To act on one or more specific JMS servers, select the check box next to the JMS server name.
      - To act on all JMS servers, select the check box next to the JMS servers title of the table.
   b. Click one of the actions displayed to change the status of the JMS servers; for example, click **Stop** to stop a JMS server.

The status of the JMS servers that you have acted on is updated to show the result of your actions.

3. **Optional:** If you want to change the properties of a JMS server, complete the following steps:
   a. Click the name of the server
   b. Set one or more of the following configuration properties:

      Initial state
      If you want the JMS server to be started automatically when the application server is next started, set this property to `Started`.

      Number of threads
      Set 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.
4. **Optional:** If you want the JMS server to host a new JMS queue, add the queue name to the Queue Names field.

   The name must match the name of a JMS Queue administrative object, including the use of upper- and lowercase.
5. **Optional:** If you want to stop the JMS server hosting a JMS queue, remove the queue name from the Queue Names field.
6. Click **OK**.
7. Save your changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the JMS 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 Managing secured applications.

1. Configure authorization settings to access JMS resources owned by the embedded WebSphere JMS provider. Authorization to access JMS resources owned by the embedded WebSphere JMS provider is controlled by settings in the *wempspath*/*wempsname*/`config/integral-jms-authorizations.xml` file.

   The settings grant or deny authenticated userids access to internal JMS provider 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.
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.

Dynamic updating is available, by ensuring proper tagging in the integral-jms-authorizations.xml file `<dyanmic-update>true</dynamic-update>`.

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 *wempspath*/*wempsname*/config/integral-jms-authorizations.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

Tuning JMS destinations

Use this task to configure the properties of a JMS destination to optimize performance of applications that use the WebSphere Application version 5 default messaging provider or WebSphere MQ as a JMS provider.

To optimize performance, configure destination properties to best fit your applications. You should also consider queue attributes of the JMS server that are associated with the queue name. For more information, see the following topics:

• “Performance considerations for WebSphere Version 5 queue destinations” on page 702

- “Performance considerations for WebSphere Version 5 topic destinations”
- “Performance considerations for WebSphere MQ queue destinations”
- “Performance considerations for WebSphere MQ topic destinations” on page 703

Performance considerations for WebSphere Version 5 queue destinations: To optimize performance, configure the queue destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued. To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

For queue destinations configured on a WebSphere Application Server version 5 node, you should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

BOQNAME
: The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

BOTHRESH
: The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

For more information about using these properties, see:

- “Handling poison messages” in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Performance considerations for WebSphere Version 5 topic destinations: To optimize performance, configure the JMS destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

For JMS destinations configured on a WebSphere Application Server version 5 node, ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:
  - SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:

- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Performance considerations for WebSphere MQ queue destinations: To optimize performance, configure the queue destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued. To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

You should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

You should also consider the queue attributes associated with the queue name you created with WebSphere MQ. Inappropriate queue attributes can reduce the performance of WebSphere operations. You can use WebSphere MQ commands to change queue attributes for the queue name.

BOQNAME
The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

BOTHRESH
The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

INDXTYPE
Set this to MSGID. This causes an index of message identifiers to be maintained, which can improve WebSphere MQ retrieval of messages.

DEFSOPT
Set this to SHARED (for shared input from the queue).

SHARE
This must be specified (so that multiple applications can get messages from this queue).

For more information about using these properties, see:

- For BOQNAME and BOTHRESH, see “Handling poison messages” in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Performance considerations for WebSphere MQ topic destinations: To optimize performance, configure the topic destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued. To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:
  - SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:

- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

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.

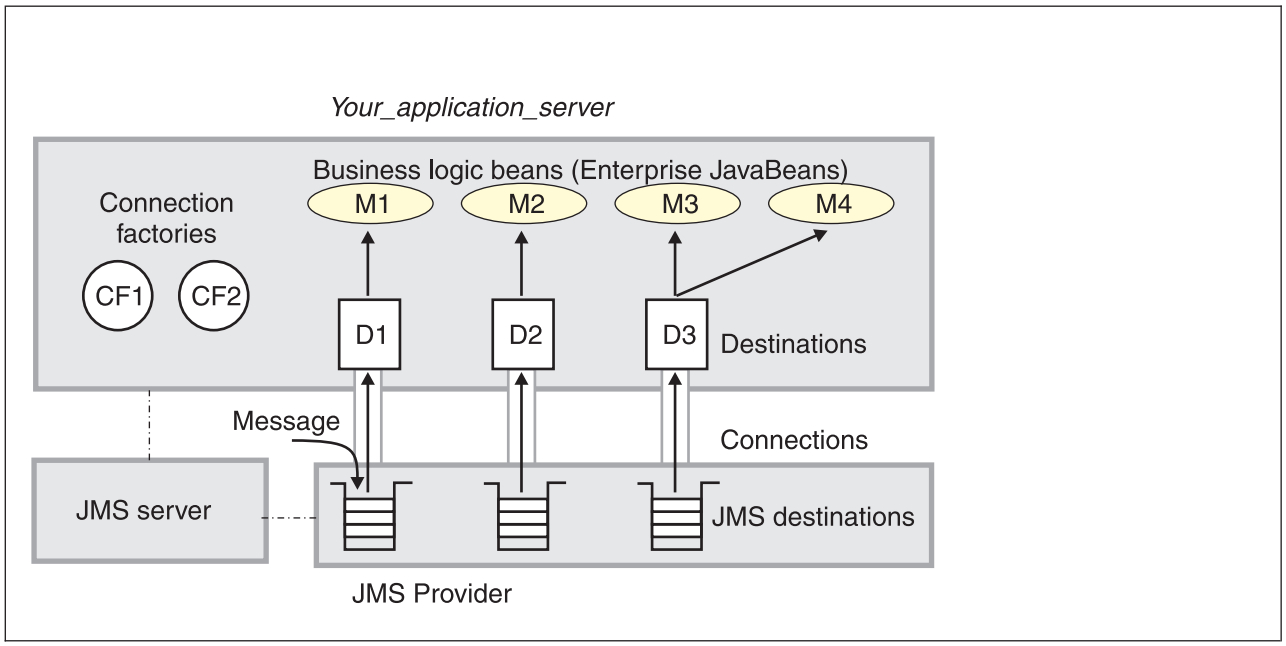


Figure 8. The main JMS components on a version 5 node. This figure shows the main JMS components on a version 5 node, from JMS provider through a connection to a destination, then to a WebSphere enterprise application (acting as a JMS client) that processes the message retrieved from the destination. For more information, see the text that accompanies this figure.

Using JMS resources of WebSphere MQ

This topic is the entry-point into a set of topics about enabling WebSphere applications to use JMS resources provided by WebSphere MQ.

You can install WebSphere MQ as a JMS provider to WebSphere Application Server. WebSphere applications can use the JMS 1.1 interfaces or JMS 1.0.2 interfaces to access JMS resources provided by WebSphere MQ, in addition to JMS resources provided by the default messaging provider (or a generic messaging provider).

You can use the WebSphere administrative console to administer the JMS connection factories and destinations provided by WebSphere MQ.

In a mixed-version WebSphere Application Server deployment manager cell, you can administer WebSphere MQ 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 WebSphere MQ as a messaging provider to WebSphere Application Server, see the following topics:

- “Installing WebSphere MQ as a JMS provider” on page 705
- “Listing JMS resources for WebSphere MQ” on page 705
- “Configuring JMS resources for the WebSphere MQ messaging provider” on page 765

Installing WebSphere MQ as a JMS provider

Use this task to install and configure WebSphere MQ with support for the Java Message Service (JMS), for use as a JMS provider to WebSphere Application Server.

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 a supported version of WebSphere MQ, see the Supported hardware and software Web page at http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html.

   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) After you install WebSphere MQ on any of the following platforms, but before you try to accept the license, apply the fix located at: http://l3.hursley.ibm.com/cgi-bin/ViewPRB.pl?1812. Otherwise, you may 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

   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. If you want to install IBM WebSphere Application Server on the same host as WebSphere MQ, and have not yet done so, install and customize WebSphere Application Server.
5. Set the MQJMS_LIB_ROOT environment variable to the directory where WebSphereMQ\Java\lib is installed. IBM WebSphere Application Server uses the MQJMS_LIB_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.

Listing JMS resources for WebSphere MQ

Use this task with the WebSphere administrative console to list the JMS resources provided by WebSphere MQ as a messaging provider.

You can use the WebSphere administrative console to display lists of the following types of JMS resources provided by WebSphere MQ. 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 WebSphere MQ, complete the following general steps:

1. Start the WebSphere administrative console.
2. In the navigation pane, click **Resources** → **JMS Providers** → **WebSphere MQ**
3. 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.
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.

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 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.
- 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 global 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 global 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.

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.

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 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-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, 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 global 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 global 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 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, BINDINGS is more desirable than CLIENT.

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.

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-3.ibm.com/software/ts/mqseries/library/manualsa/manuals/platspecific.html, 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 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.

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; 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

Use 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.

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.

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 global 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 global 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.

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.

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://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas010p.htm#HDRDCDN.

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.

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/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
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.

A value greater than or equal to zero, and less than or equal to 999 999 999.

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, as described in Installing and configuring WebSphere MQ as the JMS provider.

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” on page 765.

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 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 Queue Connection Factories** This displays a table listing any existing JMS queue connection factories, with a summary of their properties.
4. 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.
5. **Optional:** Change properties for the queue 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 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 765.

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 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 Topic Connection Factories** This displays a table listing any existing JMS topic connection factories, with a summary of their properties.
4. 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.

5. **Optional:** Change properties for the topic 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 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 optimize performance, configure the queue destination properties to best fit your applications. You should also consider queue attributes of the internal JMS server that are associated with the queue name. For more information, see “Performance considerations for WebSphere MQ queue destinations” on page 702.

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 Providers** → **WebSphere MQ**.
2. **Optional:** Change the **Scope** setting to the level at which the JMS destination is visible to applications.
3. In the contents pane, under Additional Properties, click **WebSphere MQ queue destinations** This displays a table listing any existing JMS queue destinations, with a summary of their properties.
4. 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.

5. **Optional:** Change properties for the queue destination, according to your needs.
6. **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.
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 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 optimize performance, configure the topic destination properties to best fit your applications. For more information, see "Performance considerations for WebSphere MQ topic destinations" on page 703.

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 Providers** → **WebSphere MQ**.
2. **Optional:** Change the **Scope** setting to the level at which the JMS destination is visible to applications.
3. In the contents pane, under Additional Properties, click **WebSphere MQ topic destinations** This displays a table listing any existing JMS topic destinations, with a summary of their properties.
4. 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.

5. **Optional:** Change properties for the topic 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.

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.

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 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.

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"
- "Displaying administrative lists of generic messaging resources" on page 772
- "Configuring JMS resources for a generic messaging provider" on page 778

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 778.

Displaying administrative lists of generic 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.

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 global 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.

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 global 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.

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, 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 messaging provider or WebSphere MQ).

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 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 connection factories** This displays a table listing any existing JMS connection factories, with a summary of their properties.
4. 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.
5. **Optional:** Change properties for the JMS connection factory, 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.

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 782
- 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.

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.

To view this administrative console page, click **Resources** >**Resource Adapters** > *resource_adapter* > **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

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.

To view this administrative console page, click **Resources** >**Resource Adapters** > *resource_adapter* > **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

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).

To view this administered console page, click **Resources** >**Resource Adapters** > *resource_adapter* > **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

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).

To view this administrative console page, click **Resources** >**Resource Adapters** > *resource_adapter* > **J2C Administered Objects** > *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

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, use the following task descriptions:

- Configuring the message listener service
- Adding a new listener port
- Configuring a listener port
- Deleting a listener port
- Configuring security for EJB 2.0 message-driven beans
- Administering listener ports

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 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. For more information about the property fields for listener ports, see Listener port properties.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Messaging] Message Listener Service** → **Listener Ports**

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.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
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.

Data type	Integer
Units	Seconds
Default	10
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.

Configuring security for message-driven beans that use listener ports:

Use this task to configure resource security and security permissions for Enterprise JavaBeans (EJB) Version 2.0 message-driven beans deployed to use listener ports.

Messages arriving at a listener port have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. 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 more information about configuring security for your application, see Assembling secured applications.

JMS connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define a J2C container-managed alias on the JMS connection factory definition that the message-driven bean is using to listen upon (defined by the **Connection factory JNDI name** property of the listener port). If defined, the listener uses the container-managed authentication alias for its JMSConnection security credentials instead of any application-managed alias. To set the container-managed alias, use the administrative console to complete the following steps:

1. To display the listener port settings, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Listener Ports** → *listener_port*
2. To get the name of the JMS connection factory, look at the **Connection factory JNDI name** property.
3. Display the JMS connection factory properties. For example, to display the properties of a WebSphere queue connection factory provided by the default messaging provider, click **Resources** → **JMS Providers** → **Default Messaging Provider** → → **[Content pane] WebSphere Queue Connection Factories** → *connection_factory*
4. Set the **Authentication alias** property.
5. Click **OK**

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 9. 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 following 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.

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.mai.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 Providers** in the console navigation tree. Then, click **Mail Provider** > *mail_provider* > **Mail Session**.
   c. Click the *mail_provider* or *mail_session* that you want to modify. To remove a mail provider or mail session, click **Remove** after making your selection.
   d. Click **Apply** or **OK**.
   e. 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 1.2, 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 using the following Sun licensed packages:
- `mail.jar` - Contains the JavaMail APIs, and the SMTP, IMAP, and POP3 service providers.
- `activation.jar` - Contains the JavaBeans Activation Framework.

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. See the article, *Mail: Resources for learning*, for a link to this documentation.

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.

Mail migration tip

Parts of the JavaServer Page (JSP) 1.2 specification change the way the EmailBean class works with Email.jsp.

The specifications state that the JSP container creates a JSP page implementation class for each JSP page. The name of the JSP page implementation class is implementation-dependent. The JSP page implementation object belongs to an implementation-dependent named package which can vary between one JSP and another; therefore minimal assumptions should be made. The unnamed package should not be used without explicit import of the class.

Following these specifications, you should place EmailBean.class in a package referred to it by the fully qualified packageName in Email.jsp. Otherwise, Email.jsp is unable to find EmailBean.class.

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.jar and activation.jar files. WebSphere Application Server uses the general-purpose JavaMail configuration files contained in the mail.jar and activation.jar files and does not put any mail configuration files in <user.home>and <java.home>/lib. File read permission for both the mail.jar and activation.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}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
permission java.io.FilePermission "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.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}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
permission java.io.FilePermission "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
permission java.io.FilePermission "${java.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 Version 6.0 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 Providers** in the console navigation tree.
   c. Click *URL_provider*>**URLs**.
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 Providers** in the console navigation tree.
   c. Click **URL Provider** > **URLs**.
   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 create new URL providers to handle URL protocols that are not supported by the IBM Developer Kit For the Java™ Platform. You also have the option of selecting the default URL provider, which uses the URL support provided by the kit. Any URL resource with protocols based on Java 2 Standard Edition 1.3.1, such as HyperText Transfer Protocol (HTTP) or File Transfer Protocol (FTP), can use the default URL provider.

To view this administrative console page, click **Resources** > **URL Providers**.

Name:

Specifies the administrative name for the URL provider.

Description:

Describes the URL provider for your administrative records.

URL provider settings

Use this page create new URL providers.

To view this administrative console page, click **Resources** > **URL Providers** > *URL_provider*.

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 configuration collection

Use this page to view existing Uniform Resource Locator (URL) configurations, as well as begin configuring new URLs that point to electronically accessible resources (such as directory files on a machine in a network, or a document stored in a database).

To view this administrative console page, click **Resources** > **URL Providers** > *URL_provider* > **URLs**.

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.

URL configuration settings

Use this page to configure Uniform Resource Locators (URLs) that point to electronically accessible resources, such as a directory file on a machine in a network, or a document stored in a database.

To view this administrative console page, click **Resources** > **URL Providers** > *URL_provider* > **URLs** > *URL*.

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.

Spec:

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 a environment resource factory. Begin by clicking **Resources** >**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 805 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 Providers** > *your_resource_environment_provider* > **Referenceables**. Click **New** to begin the configuration process. See the "Referenceables settings" on page 806 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 the resource environment providers.

To view this administrative console page, click **Resources** >**Resource Environment Providers**.

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 provider settings:

Use this page to create settings for a resource environment provider.

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource environment 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 Providers** > **New**.

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 entry settings

Use this page to view resource environment entries.

An environment resource can be of any arbitrary type. See the latest EJB specification for more information about resource environment references and environment resources.

To view this administrative console page, click **Resources** >**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

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 set resource environment entries, which define configuration for an environment resource that is the binding target for a resource-environment-reference in some application's deployment descriptor.

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource_environment_provider* > **Resource Environment Entries** > *resource_environment_entry*.

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 that holds the factoryClass object name of the factory that converts information in the name space into a class instance for the type of resource desired, and for the class name of the type to be returned.

Data type Drop-down menu

Referenceables collection

Use this page to specify the class name of the factory that will convert information in the name space into a class instance for the type of resource desired.

To view this administrative console page, click **Resources** >**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 for the type of resource desired

To view this administrative console page, click **Resources** >**Resource Environment Providers** > *resource_environment_provider* > **Referenceables** > *referenceable*.

Factory Classname:

Specifies a javax.naming.ObjectFactory implementation class name

Data type String

Classname:

Specifies the Java type to which a Referenceable provides access, for binding validation and to create the 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 Providers**.
3. Create the mail provider.
   a. Click **New**.
   b. Type the name of the mail provider in the Name field.
   c. Click **Apply** or **OK**.
4. 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.
5. 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**.
6. 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 begin implementing mail capabilities by selecting a JavaMail service provider, also known simply as a *mail provider*. The mail provider encapsulates a collection of protocol providers.

To view this administrative console page, click **Resources** > **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 by selecting **New** and typing values for the following settings:

Name:

Specifies the name of the JavaMail resource provider.

Description:

Specifies the resource provider description.

Mail provider settings

Use this page to establish and edit settings for your JavaMail service provider (also simply called a *mail provider*). The mail provider informs the application server of the protocols needed for your mail application.

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider* (or **Resources** > **Mail Providers** >**New**).

Supply appropriate values for the following general mail provider settings. After clicking **Apply**, select **Protocol providers** to designate the protocols for your mail system, or select **Mail sessions** to begin configuring mail sessions.

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 configure protocol provider settings.

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider* > **Protocol Providers** > *protocol_provider*.

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**.

Mail session collection

Use this page to view mail sessions that are defined under the parent mail provider, or to configure new mail sessions.

To view this administrative console page, click **Resources** > **Mail Providers** > *mail_provider* > **Mail Sessions**.

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 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.

To view this administrative console page, click **Resources**> **Mail Providers** > *mail_provider* > **Mail Sessions** > *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.

JavaMail system properties

Use this information to set custom Java virtual machine (JVM) system properties that provide additional character encoding options for your JavaMail implementation.

To view the appropriate administrative console page for setting these properties, click **Servers > Application Servers >***server_name* **> Process Definition > Java Virtual Machine > Custom Properties > New**.

To specify a custom property:

1. On the settings page, enter the property you want to configure in the Name field and the value you want to set it to in the Value field.
2. In the Description field, enter identifying information about the name-value pair.
3. Click **Apply** or **OK**.
4. Click **Save** on the console taskbar to save your configuration changes.

The following list contains the JVM system properties that you can use to customize your JavaMail application:

mail.mime.charset:

Changes the default character set for the text of JavaMail messages. Generally the character set for sending messages is the same character set that is used by the system file schema. You can use this property to specify a different character set for JavaMail messages.

Data type	String

For more information, see the latest JavaMail specification.

mail.mime.decodetext.strict:

Specifies whether or not your JavaMail application attempts to decode mail message text that does not adhere to the encoding standards set by the Internet Architecture Board in the specification document RFC 2047. (In particular, Japanese mailers might not adhere to the specification, resulting in the disruption of words by encoded text.) Set this property to `false` if you want your JavaMail component to decode message text that does not comply with RFC 2047.

Data type	Boolean
Default	true

For more information, see the latest JavaMail specification.

mail.mime.encodeeol.strict:

Specifies whether or not your JavaMail application interprets the character pairs CR and LF as line terminators when they are displayed in the part of a mail message that is not of MIME text type. Set this property to `true` if you want the JavaMail component to interpret the character pairs as line terminators, and to encode the entire portion of the message in which the characters appear.

Data type	Boolean
Default	false

For more information, see the latest JavaMail specification.

Configuring mail, URLs, and resource environment entries with scripting

This topic contains the following tasks:

Configuring new mail providers using 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 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

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

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

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

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

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

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

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

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

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

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

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

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.

Security

Securing applications and their environments

WebSphere Application Server supports the Java 2 Platform, Enterprise Edition (J2EE) model for creating, assembling, securing, and deploying applications. This article provides a high-level description of what is involved in securing resources in a J2EE environment. Applications are often created, assembled, and deployed in different phases and by different teams.

Consult the J2EE specifications for complete details.

1. Plan to secure your applications and environment. For more information, see “Planning to secure your environment” on page 826. Complete this step before you install the WebSphere Application Server.
2. Consider pre-installation and post-installation requirements. For more information, see “Implementing security considerations at installation time” on page 839. For example, during this step, you learn how to protect security configurations after you install the product.
3. Migrate your existing security systems.
4. Develop secured applications. For more information, see Developing secured applications.
5. Assemble secured applications. For more information, see Assembling secured applications. Development tools, such as the Assembling applications, are used to assemble J2EE modules and to set the attributes in the deployment descriptors.

Most of the steps in assembling J2EE applications involve deployment descriptors; deployment descriptors play a central role in application security in a J2EE environment.

Application assemblers combine J2EE modules, resolve references between them, and create from them a single deployment unit, typically an Enterprise Archive (EAR) file. Component providers and application assemblers can be represented by the same person but do not have to be.

6. Deploy secured applications. For more information, see “Deploying secured applications” on page 1246.

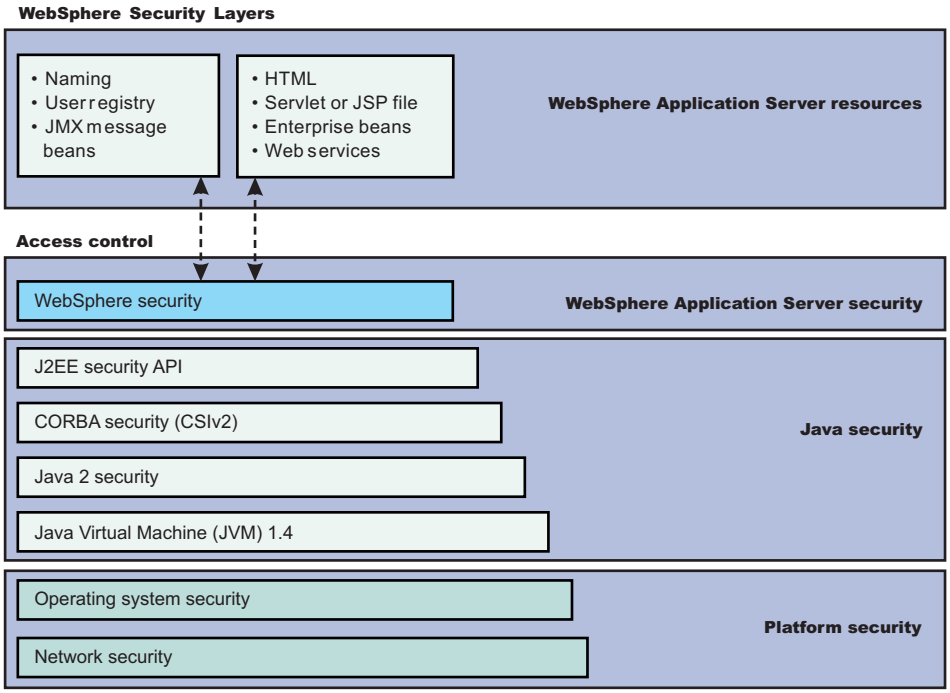
   One of the important tasks the deployer performs is mapping actual users and groups to application roles. For zSAS authorization, user or group to role mapping is done by the security administrator (through permission to a SAF EJBROLE representing the application role).
7. Test secured applications. For more information, see Testing security.
8. Manage security configurations. For more information, see “Administering security” on page 880.
9. Improve performance by tuning security configurations. For more information, see Tuning security configurations.
10. Troubleshoot security configurations. For more information, see Troubleshooting security configurations.

Your applications and production environment are secured.

See “Security: Resources for learning” on page 879 for more information on the WebSphere Application Server security architecture.

Planning to secure your environment

When you access information on the Internet, you connect through Web servers and product servers to the enterprise data at the back end. This section examines some typical configurations and common security practices. WebSphere Application Server security is built on a layered security architecture as showed in the following figure. This section also examines the security protection that is offered by each security layer and common security practice for good quality of protection in end-to-end security. The following figure illustrates the building blocks that comprise the operating environment for security within WebSphere Application Server:



- **Operating System Security** -

  The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. The operating system identity of the servant task, as established by the STARTED profile, is the identity that is used to control access to system resources such as files or sockets. For additional access protection to these resources, Java 2 security is required.

  On the z/OS platform, in addition to knowledge of Secure Sockets Layer (SSL) and Transport Layer Security (TLS), the administrator must be familiar with System Authorization Facility (SAF) and a z/OS Security Server such as Resource Access Control Facility (RACF). Using RACF, an administrator can:
  - Identify and verify users
  - Protect user and group resources at the operating system level
  - Assign identities to the started tasks for WebSphere Application Server
  - Utilize the z/OS Security Server facilities for authentication and mapping of network clients to SAF such as errors authentication and X.509 client certificates
  - Record and analyze (audit) security information

  In addition to these tasks, if the local OS user registry or SAF authorization is selected, you can use operating system security for authentication and authorization to Java 2 Platform, Enterprise Edition (J2EE) resources.
- **Network Security** - The Network Security layers provide transport level authentication and message integrity and confidentiality. You can configure the communication between separate application servers to use Secure Sockets Layer (SSL). Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.

  WebSphere Application Server z/OS provides SystemSSL for communication using the Internet. SystemSSL is composed of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS), which enable secure file transfer by providing data privacy and message integrity.
- **JVM 1.4** - The JVM security model provides a layer of security above the operating system layer.
- **Java 2 Security** - The Java 2 Security model offers fine-grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource.
- **CSIv2 Security** - CSIv2 is an IIOP-based, three-tiered, security protocol developed by the Object Management Group (OMG). This protocol provides message protection, interoperable authentication, and delegation. The three layers include a base transport security layer, a supplemental client authentication layer, and a security attribute layer. WebSphere Application Server for z/OS supports CSIv2, conformance level 0.
- **OMG CSIv2 Security** - Any calls made among secure Object Request Brokers (ORB) are invoked over the Common Security Interoperability Version 2 (CSIv2) security protocol that sets up the security context and the necessary quality of protection. After the session is established, the call is passed up to the enterprise bean layer. For backward compatibility, WebSphere Application Server supports the Secure Authentication Service (SAS) security protocol, which was used in prior releases of WebSphere Application Server and other IBM products.

  SAF authorization is an alternative to WebSphere Application Server authorizations.
- **J2EE Security** - The security collaborator enforces Java 2 Platform, Enterprise Edition (J2EE)-based security policies and supports J2EE security APIs.
- **WebSphere Security** - WebSphere Application Server security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and JMX administrative resources. It consists of WebSphere Application Server security technologies and features to support the needs of a secure enterprise environment.

WebSphere Application Server Network Deployment installation: The following figure shows a typical multiple-tier business computing environment for a WebSphere Application Server Network Deployment installation.

Important: There is a node agent instance on every computer node.

Each product application server consists of a Web container, an EJB container, and the administrative subsystem.

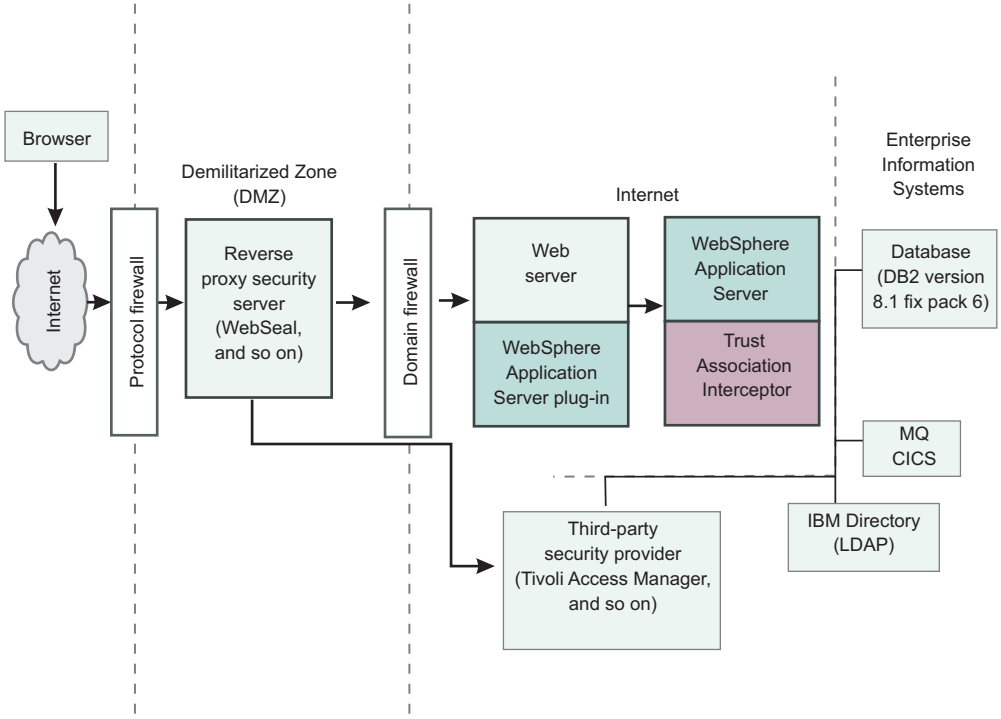
The WebSphere Application Server deployment manager contains only WebSphere administrative code and the administrative console.

The administrative console is a special J2EE Web application that provides the interface for performing administrative functions. WebSphere Application Server configuration data is stored in XML descriptor files, which must be protected by operating system security. Passwords and other sensitive configuration data can be modified using the administrative console. However, you must protect these passwords and sensitive data. For more information, see “Protecting plain text passwords” on page 840.

When using SAF registries and ICSF encryption, the requirement to store passwords in configuration data is generally avoided.

The administrative console Web application has a setup data constraint that requires the administrative console servlets and JSP files to be accessed only through an SSL connection when global security is enabled.

During installation, the administrative console is configured to use a System SSL port with a keyring that you define. The customization dialogs provide RACF customization jobs to create unique server certificates (for servers within a given cell) using a common certificate authority. It is more secure if you first enable global security and complete other configuration tasks after global security is enforced.



Global and administrative security:

WebSphere Application Servers interact with each other through CSIv2 and z/OS Secure Authentication Services (z/SAS) security protocols as well as the HTTP and HTTPS protocols.

You can configure these protocols to use Secure Sockets Layer (SSL) when you enable WebSphere Application Server global security. The WebSphere Application Server administrative subsystem in every server uses Simple Object Access Protocol (SOAP) Java Management Extensions (JMX) connectors and Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) JMX connectors to pass administrative commands and configuration data. When global security is disabled, the SOAP JMX connector uses the HTTP protocol and the RMI/IIOP connector uses the TCP/IP protocol. When global

security is enabled, the SOAP JMX connector always uses the HTTPS protocol. When global security is enabled, you can configure the RMI/IIOP JMX connector to either use SSL or to use TCP/IP. It is recommended that you enable global security and enable SSL to protect the sensitive configuration data.

Note: With APAR PQ83540 support, you can enable HTTPS for applications even when global security is disabled. You can configure the SSL port for a particular server by adding the SSL port to the HTTP port list in the server Web container in addition to where it is added to the virtual hosts in the Environment configuration. You can then connect to the Web application using HTTPS and the correct port. Internal WebSphere Application Server for z/OS communication does not use SSL unless you enable global security.

Global security and administrative security configuration is at the cell level.

When global security is enabled, you can disable application security at each individual application server by clearing the **Enable global security** option on the global security panel. The global security panel is accessed through the administrative console by clicking **Security > Global security**. Disabling application server security does not affect the administrative subsystem in that application server, which is controlled by the global security configuration only. Both administrative subsystem and application code in an application server share the optional per server security protocol configuration. For more information, see "Configuring server security" on page 902.

Security for J2EE resources: Security for J2EE resources is provided by the Web container and the EJB container. Each container provides two kinds of security: declarative security and programmatic security.

In declarative security, an application security structure includes network message integrity and confidentiality, authentication requirements, security roles, and access control. Access control is expressed in a form that is external to the application. In particular, the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. WebSphere Application Server maintains J2EE security policy, including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At run time, the container uses the security policy that is defined in the XML descriptor files to enforce data constraints and access control.

When declarative security alone is not sufficient to express the security model of an application, you might use Programmatic login to make access decisions. When global security is enabled and application server security is not disabled at the server level, J2EE applications security is enforced. When the security policy is specified for a Web resource, the Web container performs access control when the resource is requested by a Web client. The Web container challenges the Web client for authentication data if none is present according to the specified authentication method, ensures the data constraints are met, and determines whether the authenticated user has the required security role. The Web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested servlet or JavaServer Pages (JSP) file if it has one of the required security roles. Servlets and JSP pages can use the `HttpServletRequest` methods `isUserInRole` and `getUserPrincipal`.

When cell level security is enabled, unless server security is disabled, the EJB container enforces access control on EJB method invocation.

The authentication takes place regardless of whether method permission is defined for the specific EJB method. The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB code can use the `EJBContext` methods `isCallerInRole` and `getCallerPrincipal`. Use the J2EE role-based access control to protect valuable business data from access by unauthorized users from both the Internet and the intranet. Refer to Securing Web applications using an assembly tool and Securing enterprise bean applications.

Role-based security: WebSphere Application Server extends the security, role-based access control to administrative resources including the JMX system management subsystem, user registries, and JNDI name space. WebSphere administrative subsystem defines four administrative security roles:

Monitor role
A monitor can view the configuration information and status, but cannot make any changes.

Operator role
An operator can trigger run-time state changes, such as start an application server or stop an application, but cannot make configuration changes.

Configurator role
A configurator can modify the configuration information, but cannot change the state of the run time.

Administrator role
An operator as well as a configurator, which additionally can modify sensitive security configuration and security policy such as setting server ID and password, enable or disable global security and Java 2 security, and map users and groups to the administrator role.

A user with the configurator role can perform most administrative work including installing new applications and application servers. There are certain configuration tasks a configurator does not have sufficient authority to do when global security is enabled, including modifying a WebSphere Application Server identity and password, LTPA password and keys, and assigning users to administrative security roles. Those sensitive configuration tasks require the administrative role because the server ID is mapped to the administrator role.

Those sensitive configuration tasks require the administrative role.

WebSphere Application Server administrative security is enforced when global security is enabled. It is recommended that WebSphere Application Server global security be enabled to protect administrative subsystem integrity. Application server security can be selectively disabled if there is no sensitive information to protect. For securing administrative security, refer to “Assigning users to administrator roles” on page 910 and “Assigning users and groups to roles” on page 1247.

Java 2 security permissions: WebSphere Application Server uses the Java 2 security model to create a secure environment to run application code. Java 2 security provides a fine-grained and policy-based access control to protect system resources such as files, system properties, opening socket connections, loading libraries, and so on. The J2EE Version 1.4 specification defines a typical set of Java 2 security permissions that Web and EJB components expect to have. These permissions are shown in the following table.

Table 6. Java 2 security permissions set for EJB components

Security Permission	Target	Action
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.util.PropertyPermission	*	read

The WebSphere Application Server Java 2 security default policies are based on the J2EE Version 1.4 specification. The specification granted Web components read and write file access permission to any file in the file system, which might be too broad. The WebSphere Application Server default policy gives Web components read and write permission to the subdirectory and the subtree where the Web module is installed. The default Java 2 security policy for all Java virtual machines and WebSphere Application Server processes are contained in the following policy files:

`${java.home}/jre/lib/security/java.policy`
Used as the default policy for the Java virtual machine (JVM).

$*WAS_HOME*/properties/server.policy
Used as the default policy for all product server processes

To simplify policy management, WebSphere Application Server policy is based on resource type rather than code base (location). The following files are the default policy files for WebSphere Application Server subsystem. These policy files, which are an extension of WebSphere Application Server run time and are referred to as *Service Provider Programming Interfaces (SPI)*, are shared by multiple J2EE applications:

$*WAS_HOME*/config/cells/*cell_name*/nodes/*node_name*/spi.policy
Used for embedded resources that are defined in the `resources.xml` file, such as the Java Message Service (JMS), JavaMail, and JDBC drivers.

$*WAS_HOME*/config/cells/*cell_name*/nodes/*node_name*/library.policy
Used by the shared library that is defined by the WebSphere Application Server administrative console.

$*WAS_HOME*/config/cells/*cell_name*/nodes/*node_name*/app.policy
Used as the default policy for J2EE applications.

In general, applications should not require more permissions to run than those recommended by the J2EE specification to be portable among various application servers. However, some applications might require more permissions. WebSphere Application Server supports a per application policy file, `was.policy`, to be packaged together with each application from granting extra permissions to that application.

Attention: Grant extra permissions to an application after careful consideration because of the potential of compromising the system integrity.

Loading libraries into the WebSphere Application Server does allow applications to leave the Java sandbox. When you install an application for WebSphere Application Server, the server uses a permission filtering policy file to alert you when an application requires additional permissions and causes the affected application installation to fail. For example, it is recommended that you not give the `java.lang.RuntimePermission` *exitVM* permission to an application so that application code cannot terminate WebSphere Application Server. The filtering policy is defined by the `filterMask` in `${WAS_INSTALL_ROOT}/profiles/`*profile_name*`/config/cells/`*cell_name*`/filter.policy`. Moreover, WebSphere Application Server also performs run-time permission filtering that is based on the run-time filtering policy to ensure that application code is not granted a permission that is considered harmful to system integrity.

Therefore, many applications developed for prior releases of WebSphere Application Server might not be Java 2 Security ready. To migrate those applications to WebSphere Application Server Version 6.0.x quickly, you might temporarily give those applications `java.security.AllPermission` in the `was.policy` file. It is recommended that you test those applications to ensure that they execute in an environment where Java 2 Security is active. For example, identity what extra permissions, if any, are required, and to grant only those permissions to a particular application. Not granting applications `AllPermission` can certainly reduce the risk of compromising system integrity. For more information on migrating applications to WebSphere Application Server Version 6.0.x, refer to “Migrating Java 2 security policy” on page 1241.

The WebSphere Application Server run time uses Java 2 Security to protect sensitive run-time functions; therefore, it is recommended that you enforce Java 2 security. Applications that are granted `AllPermission` not only have access to sensitive system resources, but also WebSphere Application Server run-time resources and can potentially cause damage to both. In cases where an application can be trusted to be safe, WebSphere Application Server allows Java 2 Security to be disabled on a per application server basis. You can enforce Java 2 security by default in the security center and disable the per application server Java 2 Security flag to disable it at the particular application server.

When you specify the **Enable global security** and **Enable Java 2 Security** options on the Global security panel of the administrative console, the information, along with other sensitive configuration data, are

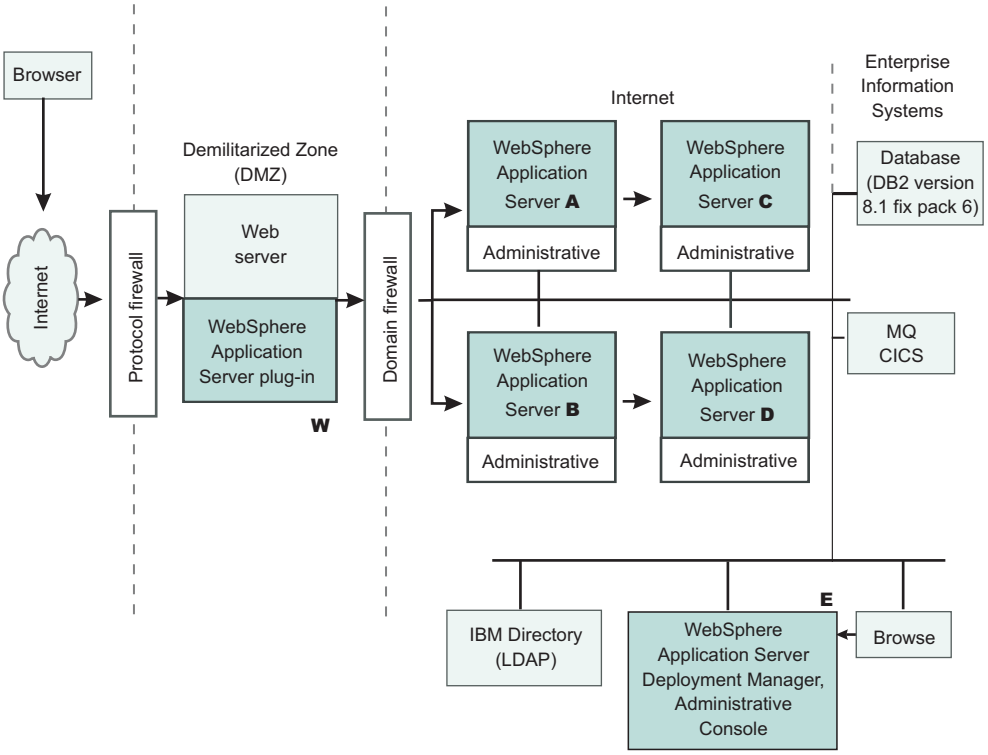
stored in a set of XML configuration files. Both role-based access control and Java 2 Security permission-based access control are employed to protect the integrity of the configuration data. The example uses configuration data protection to illustrate how system integrity is maintained.

- When Java 2 security is enforced, the application code cannot access the WebSphere Application Server run-time classes that manage the configuration data unless it is granted the required WebSphere Application Server run-time permissions.
- When Java 2 security is enforced, application code cannot access the WebSphere Application Server configuration XML files unless it has been granted the required file read and write permission.
- The JMX administrative subsystem provides SOAP over HTTP or HTTPS and RMI/IIOP remote interface to enable application programs to extract and to modify configuration files and data. When global security is enabled, an application program can modify the WebSphere Application Server configuration if the application program has presented valid authentication data and the security identity has the required security roles.
- If a user can disable Java 2 security, then that user can modify the WebSphere Application Server configuration including the WebSphere Application Server security identity and authentication data along with other sensitive data. Only users with the administrator security role can disable Java 2 security.
- Only users with administrator role can disable global security, change server ID and password, map users and groups to administrative roles, and so on.

The CSIv2 security protocol also supports client certificate authentication. SSL client authentication can also be used to set up secure communication among selected set of servers based on trust relationship.

If you start from the WebSphere Application Server plug-in at the Web server, SSL mutual authentication can be configured between it and the WebSphere Application Server HTTPS server. When using self signed certificates, one can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Server servers as shown in the following picture. Suppose you want to restrict the HTTPS server in WebSphere Application Server **A** and in WebSphere Application Server **B** to accept secure socket connections only from WebSphere Application Server plug-in **W**.

You can generate three self-signed certificates using RACF, such as certificate **W, A**, and **B**. The WebSphere Application Server plug-in is configured to use certificate **W** and trust certificate **A** and **B**. The HTTPS server of WebSphere Application Server **A** is configured to use certificate **A** and to trust certificate **W**. The HTTPS server of WebSphere Application Server **B** is configured to use certificate **B** and to trust certificate **W**.



The trust relationship depicted in the previous picture is shown in the following table.

Server	Key	Trust
WebSphere Application Server plug-in	W	A, B
WebSphere Application Server A	A	W
WebSphere Application Server B	B	W

In a z/OS installation, the WebSphere Application Server deployment manager is a central point of administration. System management commands are sent from the deployment manager to each individual application server. When global security is enabled, all WebSphere Application Server servers can be configured to require SSL and mutual authentication. Suppose you want to further restrict that WebSphere Application Server application. Server **A** can only communicate with WebSphere Application Server **C** and WebSphere Application Server **B** can only communicate with WebSphere Application Server **D**. Note that as mentioned previously, all WebSphere Application Servers must be able to communicate with WebSphere Application Server deployment manager **E**. Therefore, when using self-signed certificates, you might set up the CSIv2 and SOAP/HTTPS key and trust relationship as shown in the following table.

Server	Key	Trust
WebSphere Application Server A	A	C, E
WebSphere Application Server B	B	D, E
WebSphere Application Server C	C	A, E
WebSphere Application Server D	D	B, E
WebSphere Application Server Deployment Manager E	E	A, B, C, D

When WebSphere Application Server is configured to use an Lightweight Directory Access Protocol (LDAP) user registry, you also can configure SSL with mutual authentication between every application

server and the LDAP server with self-signed certificates so that a password is not passed in clear text from WebSphere Application Server to the LDAP server.

In this example, the node agent processes are not discussed. Each node agent must communicate with application servers on the same node and with the Deployment Manager. Node agents also must communicate with LDAP servers when they are configured to use an LDAP user registry. It is reasonable to let the deployment manager and the node agents use the same certificate. Suppose application server **A** and **C** are on the same computer node. The Node agent on that node needs to have certificates **A** and **C** in its trust store.

1. Determine which versions of WebSphere Application Server you are using.
2. Review the WebSphere Application Server security architecture.
3. Review each of the following topics as also defined in Related reference.
   - "Global security and server security" on page 899
   - "Authentication protocol for EJB security" on page 1154
     - "Supported authentication protocols" on page 1161
     - "Common Secure Interoperability Version 2 features" on page 1157
     - "Identity assertion" on page 1158
   - "Authentication mechanisms" on page 915
     - "Lightweight Third Party Authentication settings" on page 922
     - "Trust associations" on page 924
     - "Single signon" on page 930
   - "User registries" on page 951
     - "Local operating system user registries" on page 956
     - "Lightweight Directory Access Protocol" on page 960
   - "Custom user registries" on page 974
   - "Java 2 security" on page 1208
     - "Java 2 security policy files" on page 1217
   - "Java Authentication and Authorization Service" on page 1002
     - Programmatic login
   - "J2EE Connector security" on page 1032
   - "Access control exception" on page 1213
     - Role-based authorization
     - "Administrative console and naming service authorization" on page 907
   - "Secure Sockets Layer" on page 1179
     - "Authenticity" on page 1181
     - "Confidentiality" on page 1181
     - "Integrity" on page 1183

Security considerations when adding a Base Application Server node to Network Deployment

At some point, you might decide to centralize the configuration of your stand-alone base application servers by adding them into a Network Deployment cell. If your base application server is currently configured with security, there are some issues to consider. The major issue when adding a node to the cell is whether the user registries between the base application server and the Deployment Manager are the same.

When adding a node to a cell, the newly federated node automatically inherits the user registry (LocalOS, LDAP or Custom), authentication mechanism (LTPA or ICSF), and authorization setting (WebSphere bindings or System Authorization Facility (SAF) EJBROLE profiles) of the existing Network Deployment cell.

For distributed security, all servers in the cell must use the same user registry and authentication mechanism. To recover from a user registry change, you must modify your applications so that the user and group to role mappings are correct for the new user registry. To do this, see the article on “Assigning users and groups to roles” on page 1247.

Another important consideration is the SSL public-key infrastructure. Prior to performing `addNode` with the Deployment Manager, verify that `addNode` can communicate as an SSL client with the Deployment Manager. This requires that the addNode truststore (configured in `sas.client.props`) contains the signer certificate of the Deployment Manager personal certificate as found in the keystore (specified in the administrative console).

The following are other issues to consider when running the `addNode` command with security:

1. When attempting to run system management commands such as `addNode`, you need to explicitly specify administrative credentials to perform the operation. The `addNode` command accepts `-username` and `-password` parameters to specify the userid and password, respectively. The user ID and password that are specified must be an administrative user; for example, a user that is a member of the console users with **Operator** or **Administrator** privileges or the administrative user ID configured in the User Registry. An example for `addNode`, `addNode CELL_HOST 8879 -includeapps -username user -password pass`. `-includeapps` is optional, but this option attempts to include the server applications into the Deployment Manager. The `addNode` command might fail if the user registries used by the WebSphere Application Server and the Deployment Manager are not the same. To correct this problem, either make the user registries the same or turn off security. If you change the user registries, remember to verify that the users to roles and groups to roles mappings are correct. See addNode command for more information on the addNode syntax.

   Note: You can also run the addNode command using the z/OS Customization Dialog. If you issue the addNode command with security enabled using the z/OS Customization Dialog or command line, you must use a user ID with authority and specify the `-user` and `-password` options.
2. Adding a secured remote node through the administrative console is not supported. You can either disable security on the remote node before performing the operation or perform the operation from the command line using the addNode script.
3. Before running the `addNode` command, you must verify that the truststore files on the nodes communicate with the keystore files and SAF Keyring owned by the Deployment Manager and vice versa. If you have generated the certificates for deployment manager using the same certificate authority as you used for the node agent process, this will be successful. Note that the following SSL configurations must contain keystores and truststores that can interoperate:
   - System SSL repertoire specified in the Administrative Console using **System Administration > Deployment Manager > HTTP Transports > sslportno > SSL**
   - SSL repertoire for appropriate JMX Connector if SOAP is specified **System Administration > dmgr > Administration Services > JMX Connectors > SOAPConnector > Custom Properties > sslConfig**
   - SSL repertoire specified in NodeAgent **System Administration > Node agents > NodeAgent Server > Administration Services > JMX Connectors > SOAPConnector > Custom Properties > sslConfig**

   Note: WebSphere Application Server for z/OS defines security domain names using the z/OS Customization Dialog. Use caution when adding a node to a Deployment Manager configuration that defines a different security domain.
4. After running addNode, the application server is in a new SSL domain. It might contain SSL configurations that point to keystore and truststore files that are not prepared to interoperate with other servers in the same domain. Consider which servers will be intercommunicating and ensure that the servers are trusted within your truststore files.

Proper understanding of the security interactions between distributed servers greatly reduces problems encountered with secure communications. Security adds complexity because additional function needs to be managed. For security to function, it needs thorough consideration during the planning of your infrastructure. This document helps to reduce the problems that could occur due to inherent security interactions.

When you have security problems related to the WebSphere Application Server Network Deployment environment, check the Troubleshooting security configurations section to see if you can get information about the problem. When trace is needed to solve a problem, because servers are distributed, quite often it is required to gather trace on all servers simultaneously while recreating the problem. This trace can be enabled dynamically or statically, depending on the type problem occurring.

Security considerations specific to a multi-node or process Network Deployment environment

WebSphere Application Server Network Deployment allows for centralized management of distributed nodes and application servers. This inherently brings complexity, especially when security is included into the mix. Because everything is distributed, security plays an even larger role in ensuring that communications are appropriately secure between applications servers and node agents, and between node agents (a node specific configuration manager) and the Deployment Manager (a domain-wide, centralized configuration manager). The following issues should be considered when operating in this environment, but preferably prior to going to this environment.

Because the processes are distributed, an authentication mechanism must be selected that supports an authentication token such as LTPA. The tokens are encrypted and signed and therefore, forwardable to remote processes. However, the tokens have expirations. The Simple Object Access Protocol (SOAP) connector (the default connector) used for administrative security does not have retry logic for expired tokens, however, the protocol is stateless so a new token is created for each request (if there is not sufficient time to execute the request with the given time left in the token). An alternative connector is the Remote Method Invocation (RMI) connector, which is stateful and has some retry logic to correct expired tokens by resubmitting the requests after the error is detected.

Note: LTPA Tokens also have expiration times that are set on the WebSphere Application Server Administrative Console

Additional considerations are dealing with SSL.WebSphere Application Server for z/OS can use RACF keyrings to store the keys and truststores used for SSL, but different SSL protocols are used internally. You must be sure to set up both:

- A System Secure Sockets Layer (SSL) repertoire for use by the Web Container
- A Java Secure Sockets Extension (JSSE) SSL repertoire for use by the SOAP HTTP connector if the SOAP connector is used for administrative requests

Verify that the keystores and truststores you configure are setup to trust only the servers to which they communicate. But make sure they do include the necessary signer certificates from those servers in the trustfiles of all servers in the domain. When using a certificate authority (CA) to create personal certificates, it is easier to ensure that all servers trust one another by having the CA root certificate in all the signers.

The customization dialogs for WebSphere Application Server for z/OS use the same certificate authority to generate certificates for all servers within a given cell, including those of the node agents and the deployment manager.

The following are issues to consider when using or planning for a Network Deployment environment.

1. When attempting to run system management commands such as stopNode, you should explicitly specify administrative credentials to perform the operation. Most commands accept `-user` and

`-password` parameters to specify the user ID and password, respectively. The user ID and password that are specified should be an administrative user; for example, a user who is a member of the console users with **Operator** or **Administrator** privileges or the administrative user ID configured in the user registry. An example for stopNode, `stopNode -username user -password pass`.

2. Verify that the configuration at the node agents are always synchronized with the Deployment Manager prior to starting or restarting a node. To manually get the configuration synchronized, issue the syncNode command from each node that is not synchronized. To synchronize the configuration for node agents that are started, click **System Administration** > **Nodes** and select all started nodes. Click **Synchronize**.
3. Verify that the LTPA token expiration period is long enough to complete your longest downstream request. Some credentials are cached and therefore the timeout does not always count in the length of the request.
4. The administrative connector used by default for system management is SOAP. SOAP is a stateless HTTP protocol. For most situations, this connector is sufficient. When running into a problem using the SOAP connector it might be desirable to change the default connector on all servers from SOAP to RMI. The RMI connector uses CSIv2, a stateful, interoperable protocol, and can be configured to use identity assertion (downstream delegation), message layer authentication (BasicAuth or Token), and client certificate authentication (for server trust isolation). To change the default connector on a given server, go to **Administration Services** in Additional Properties for that server.
5. An error message might occur within the administrative subsystem security. This indicates that the sending process did not supply a credential to the receiving process. Typically the causes for this problem are:
   - The sending process has security disabled while the receiving process has security enabled. This typically indicates one of the two processes are not in sync with the cell.

     Note: Having security disabled for a specific application server should not have any effect on administrative security.

Proper understanding of the security interactions between distributed servers greatly reduces problems encountered with secure communications. Security adds complexity because additional function must be managed. For security to work properly, it needs thorough consideration during the planning of your infrastructure. Hopefully, this document will help to reduce the problems that can occur due to inherent security interactions.

When you have security problems related to the WebSphere Application Server Network Deployment environment, check the Troubleshooting security configurations section to find additional information about the problem. When trace is needed to solve a problem because servers are distributed, quite often it is required to gather trace on all servers simultaneously while recreating the problem. This trace can be enabled dynamically or statically, depending on the type problem occurring.

Creating login key files

1. Create a login key file. The authenticating user IDs, passwords, and target realms for each different target server are specified in the login key file, which is an ASCII file. When the security authentication service processes the login key file, the passwords in the file are encoded.
2. Add information to the login key file in the following format:

   Realm_name User_ID Password
3. Make sure that the data conforms to the following rules:
   - One realm name
   - One user ID, and one password defined in each entry
   - One entry per line
   - No blank lines between entries
   - Comments on separate lines only
   - Begin any comment with a pound sign (#):

Example:

```
# Sample key file
#
# First target realm
#
TargetRealm serverID serverPassword
#
# Second target realm
#
TargetRealm2 serverID2 serverPassword2
#
# End of key file
```

The realm name of a WebSphere Application Server for z/OS target is the IP name of the daemon, as specified in the configuration of the WebSphere Application Server for z/OS product. The user ID and password are those defined for secured WebSphere Application Server for z/OS servers.

After creating the login key files, read the article entitled, “Preparing truststore files.”

Preparing truststore files

Secure Sockets Layer (SSL) protocol protects the communication between WebSphere Application Servers. To complete the SSL connection, establish a valid truststore file for the WebSphere Application Server. A truststore is a key database file that contains the public keys. A keystore is anything that Java or the System SSL libraries can read to acquire key information. For more information about how to create a new keystore, see “Creating login key files” on page 837.

1. Extract the public key of the server by using the key management tool from WebSphere Application Server. For details, see Configuring the server for request decryption: choosing the decryption method.

   Note: For more details on using z/OS and keyrings, see “Planning to secure your environment” on page 826.
2. Add the public key from the WebSphere Application Server as a signer certificate into the requesting WebSphere Application Server truststore.

The WebSphere Application Server truststore file is now ready to use for SSL connections with the WebSphere Application Server.

See “Configuring the application server for interoperability” for information on interoperability.

Configuring the application server for interoperability

After the truststore file is ready, complete the following steps to configure the WebSphere Application Server.

1. Configure the enterprise beans that access WebSphere Application Server. Before deploying the enterprise beans, configure the RunAs Identity.
2. Enable security.
3. Enable outbound SAS authentication protocol.
4. Specify the truststore file in an Secure Sockets Layer (SSL) configuration alias and configure the WebSphere Application Server with that alias.
5. Set the **Request timeout** and **Locate request timeout** values to zero for the Object Request Broker (ORB) service.

   When the WebSphere Application Server z/OS application server first starts, no server region is available for processing work. It is therefore recommended that you set these two properties to zero to prevent potential time outs.

6. Specify a security property named com.ibm.CORBA.keyFileName for the absolute path of the login key file created earlier. This does not apply for z/OS.
7. Restart the WebSphere Application Server.

Implementing security considerations at installation time

Complete the following tasks to implement security before, during, and after installing WebSphere Application Server.

1. Installing the product and additional software This step describes how to install WebSphere Application Server on the z/OS platform.
2. During installation you are prompted to Migrating security configurations from previous releases.
3. "Securing your environment after installation." This step provides information on how to protect password information after you install WebSphere Application Server.

Securing your environment after installation

WebSphere Application Server depends on several configuration files 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.

The files in the *WAS_HOME*/config and *WAS_HOME*/properties directories 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, who perform partial WebSphere Application Server administrative tasks, like configuring, starting servers and stopping servers, need permissions as well.

The files in the *WAS_HOME*/properties directory that must be readable by everybody 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

Note: The value for *WAS_HOME* directory is specified in the customization dialogs when WebSphere Application Server for z/OS is installed (for both the base product and Network Deployment).

Secure files on WebSphere Application Server for z/OS systems.

1. Use the z/OS Customization Dialog and follow the generated instructions to customize your system.

   The customization jobs that are generated perform the following functions:

   - Create System Authorization Facility (SAF) WebSphere Application Server user IDs that are needed for WebSphere administrator and WebSphere server processes
   - Create a SAF WebSphere Application Server configuration group and add the SAF WebSphere Application Server user IDs
   - Provide a mapping from a Java 2, Enterprise Edition (J2EE) principal to SAF user ID (you can generate a sample mapping module or you can specify one that you created yourself)
   - Associate WebSphere Application Server-started tasks with the SAF user IDs and groups defined previously

- Populate the file system with the system and property files that are needed to run WebSphere Application Server
- Change the ownership of these files to that of the WebSphere Application Server administrator
- Create the appropriate file permissions

Note: All files in the *WAS_HOME*/config directory must have write and read access by all members of the WebSphere configuration group, but must not be accessible by everyone (mode 770). All files in *WAS_HOME*/properties must have write and read access by all members of the WebSphere configuration group. Set the access permissions for the following files as it pertains to your security guidelines:

- 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

For example, you might issue the following command: chmod 770 *file_name*. *file_name* is the name of the file listed previously. These files contain sensitive information such as passwords.

2. Add WebSphere administrators who perform full or partial WebSphere Application Server administration tasks to the WebSphere configuration group.
3. Restrict access to the /var/mqm directories and the log files that are needed for WebSphere Application Server embedded messaging (or WebSphere MQ as the JMS provider). Give write access only to the mqm user ID or members of the mqm user group.

After securing your environment, only the users given 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.

Protecting plain text passwords

WebSphere Application Server contains several plain text passwords. These passwords are not encrypted, but are encoded. WebSphere Application Server provides the PropFilePasswordEncoder utility, which you can use to encode these passwords. However, the utility does not encode passwords that are contained within XML or XMI files. Instead, WebSphere Application Server automatically encodes the passwords in the following XML or XMI files as the files are modified by the administrative console.

Table 7. XML and XMI files the contain plain text passwords

File name	Additional information
WAS_INSTALL_ROOT/profiles/*profile_name*/config/cells/*cell_name*/security.xml	The following fields contain encoded passwords: • **LTPA password** • **JAAS authentication data** • **User registry server password** • **LDAP user registry bind password** • **Key store password** • **Trust store password**

Table 7. XML and XMI files the contain plain text passwords (continued)

File name	Additional information
war/WEB-INF/ibm_web_bnd.xml	Specifies the passwords for the default basic authentication for the ″resource-ref″ bindings within all the descriptors (except in the Java cryptography architecture).
ejb jar/META-INF/ibm_ejbjar_bnd.xml	Specifies the passwords for the default basic authentication for the ″resource-ref″ bindings within all the descriptors (except in the Java cryptography architecture).
client jar/META-INF/ibm-appclient_bnd.xml	Specifies the passwords for the default basic authentication for the ″resource-ref″ bindings within all the descriptors (except in the Java cryptography architecture).
ear/META-INF/ibm_application_bnd.xml	Specifies the passwords for the default basic authentication for the ″run as″ bindings within all the descriptors.
WAS_INSTALL_ROOT /profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/servers/*server_name*/server.xml	The following fields contain encoded passwords: • **Key store password** • **Trust store password** • **Authentication target password** • **Session persistence password** • **DRS Client data replication password**
WAS_INSTALL_ROOT/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/servers/*server_name*/resources.xml	The following fields contain encoded passwords: • **WAS40Datasource password** • **mailTransport password** • **mailStore password** • **MQQueue queue mgr password**
For Network Deployment: *WAS_INSTALL_ROOT*/profiles/*profile_name*/config/cells/*cell_name*/ws-security.xml	
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	

You can use the PropFilePasswordEncoder utility to encode the passwords that are found in the following files.

Table 8. Files that you can encode using the PropFilePasswordEncoder utility

File name	Additional information
WAS_INSTALL_ROOT/profiles/*profile_name*/properties/sas.client.props	Specifies the passwords for the following files: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
WAS_INSTALL_ROOT/profiles/*profile_name*/properties/soap.client.props	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.SOAP.loginPassword
WAS_INSTALL_ROOT/profiles/*profile_name*/properties/sas.tools.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword

Table 8. Files that you can encode using the PropFilePasswordEncoder utility (continued)

File name	Additional information
WAS_INSTALL_ROOT/profiles/*profile_name*/properties/sas.stdclient.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
WAS_INSTALL_ROOT/profiles/*profile_name*/properties/wsserver.key	

To re-encode a password in one of the previous files, complete the following steps:

1. Access the file using a text editor and type over the encoded password in plain text. The new password is shown in plain text and must be encoded.
2. Use the `PropFilePasswordEncoder.bat` or `PropFilePasswordEncode.sh` file in the *WAS_INSTALL_ROOT*/profiles/*profile_name*/bin/ directory to re-encode the password.

   If you are re-encoding z/SAS properties files, type `PropFilePasswordEncoder` *file_name* `-sas` and the `PropFilePasswordEncoder.bat` file encodes the known z/SAS properties.

   If you are encoding files that are not z/SAS properties files, type `PropFilePasswordEncoder` *file_name password_properties_list*

   file_name is the name of the z/SAS properties file. *password_properties_list* is the name of the properties to encode within the file.

   Use the `PropFilePasswordEncoder` utility to encode WebSphere Application Server password files only. The utility cannot encode passwords contained in XML files or other files that contain open and close tags.

If you reopen the affected file or files, the passwords do not display in plain text. Instead, the passwords appear encoded. WebSphere Application Server does not provide a utility for decoding the passwords.

Note: The reliance on passwords in configuration files can be minimized on WebSphere Application Server for z/OS by taking advantage of z/OS-specific features:

- Using a SAF registry removes the requirement for a user registry server password.
- Using ICSF as the authentication mechanism moves the encryption key into the hardware.
- Selection of SAF authorization and delegation so case role-to-user binding passwords are removed.
- Trust and key file passwords are no longer required when a RACF keyring is used for all SSL repertoires.
- The need for JAAS authentication data might be removed when native connectors are used,and if sync-to-thread is configured or allowed.

Setting up WebSphere Application Server for z/OS security

WebSphere Application Server for z/OS supports access to resources by clients and servers in a distributed. Determine how to control access to these resources and prevent inadvertent or malicious destruction of the system or data.

These are the pieces in the distributed network that you must consider:

- You must authorize servers to the base operating system services in z/OS or OS/390. These services include SAF security, database management, and transaction management.
  - For the server clusters, you must distinguish between controllers and servants. Controllers run authorized system code, so they are trusted. Servants run application code and are given access to resources, so carefully consider the authorization you give servants.
  - You must also distinguish between the level of authority for run-time servers and for your own application servers have. For example, the node needs the authority to start other clusters, while your own application clusters do not need this authority.

- You must authorize clients (users) to servers and objects within servers. The characteristics of each client requires special consideration:
  - Is the client on the local system or is it remote? The security of the network becomes a consideration for remote clients.
  - Will you allow unidentified (unauthenticated) clients to access the system? Some resources on your system might be intended for public access, while others you might need to protect. To access protected resources, clients must establish their identities and have authorization to use those resources.
- *Authentication* is the process of establishing the identity of a client in a particular context. A client can be an end user, a machine, or an application. The term authentication mechanism in WebSphere Application Server on z/OS refers more specifically to the facility in which WebSphere identifies an authenticated identity, using HTTP and JMX facilities. When configuring a cell, you must select a single authentication mechanism. The choices for authentication mechanism include:
  - Simple WebSphere Authorization Mechanism (SWAM) - only on Base Application Server, not available on the Network Deployment configuration
  - Lightweight Third Party Authentication (LTPA)
  - Integrated Cryptograqphic Service Facility (ICSF)
- 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. Implementation is provided to support multiple operating system or operating environment-based user registries. When configuring a cell, you must select a single user registry. The user registry can be local or remote. The choices for user registry include:
  - SAF-based local registry (default)
  - Lightweight Directory Access Protocol (LDAP) - LDAP can be either a local or remote registry
  - Custom user registry - A custom user registy is set up to meet unique registry needs. WebSphere provides a simple user registry sample called the FileBasedRegistrySample.

If you need to protect resources, it is critical that you identify who accesses those resources. Thus, any security system requires client (user) identification, also known as authentication. In a distributed network supported by WebSphere Application Server for z/OS, clients can access resources from:
- Within the same system as a server
- Within the same sysplex as the server
- Remote z/OS or OS/390 systems
- Heterogeneous systems, such as WebSphere Application Server on distributed platforms, CICS, or other J2EE -compliant systems.

Additionally, clients can request a service that requires a server to forward the request to another cluster. In such cases, the system must handle delegation, the availability of the client identity for use by intermediate clusters and target clusters.

Finally, in a distributed network, how do you verify that messages being passed are confidential and have not been tampered? How do you verify that clients are who they claim to be? How do you map network identities to z/OS or OS/390 identities? These issues are addressed by the following support in WebSphere Application Server for z/OS:
- The use of SSL and digital certificates
- Kerberos
- Common Secure Interoperability, Version 2 (CSIv2)

Security customization dialog settings: The Customization Dialog enables you to create a security domain for your WebSphere Application Server for z/OS configuration. For more information, see the following articles:

- Planning a security domain

  The article provides the background, planning, and the variable information that is needed for configuration.

- Creating a security domain

  The article explains the process of configuring the security domain using the Customization Dialog.

Note:

- You must set up a base Application Server using the dialogs before using this one to set up a Network Deployment node, which is managed by the deployment manager process (dmgr). It is critical that you **LOAD** saved environment variables from the base Application Server into the deployment manager node that federates the base node. Do this before performing security customization on the deployment manager node.
- If the APPL class is active and you have defined a profile for WebSphere, make sure that all z/OS identities using WebSphere services have READ permission to the WebSphere Application Server APPL profile. This includes all WebSphere Application Server identities, WebSphere Application Server unauthenticated identities, WebSphere Application Server administrative identities, user IDs based on role-to-user mappings, and all user identities for system users. If you have not defined a security domain, the APPL profile used is CBS390 or the name used as the security domain identifier. If you have defined a security domain, the APPL profile used is the security domain name.
- When adding an administrator to the administrative console using local operating system security, if the APPL class is activated, the administrator's user ID must be authorized to the CBS390 (or the name specified as the security domain identifier) APPL class for RACF as well. If the administrator's user ID is not authorized to CBS390 APPL, message BBOS0108E is issued, indicating that the credential-handling function (RunAsGetSpecCred) failed in routine because the user is not authorized.

Enabling global security for WebSphere Application Server:

Before you can enable global security you must select both an authentication mechanism and a user registry.

You need to start the administrative console by specifying the following Web site:
`http://server_hostname:9060/ibm/console.`

Perform the following steps to enable global security.

1. Click **Security** > **Global Security** in the Navigation tree on the left.
2. Select the **Enable global security** option. Global security is disabled by default.
3. Select the **Enforce Java 2 Security** option to enable Java 2 Security permission checking. By default, Java 2 security is disabled. However, if you enable global security, Java 2 security is automatically enabled. You can choose to disable Java 2 security, even when global security is enabled.

   When Java 2 Security is enabled and if an application requires more Java 2 security permissions then are granted in the default policy, then 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. Review the Java 2 Security and Dynamic Policy documentation if you are unfamiliar with Java 2 security.
4. Select the **Enforce fine-grained JCA security** option if you need to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data. For more information, see "Global security settings" on page 886.
5. Select the **Use domain-qualified user IDs** option. If this option is enabled, user names appear with their fully qualified domain attribute when retrieved programmatically.
6. Enter the cache timeout value for security cache in seconds in the **Cache timeout** field. When the timeout is reached, the Application Server clears the security cache and rebuilds the security data. Since this affects performance, this value should not be set too low. Default: 600 seconds.

7. Select the **Issue permission warning** option. The `filter.policy` file contains a list of permissions that an application should not have according to the J2EE 1.3 Specification. If an application is installed with a permission specified in this policy file and this option is enabled, a warning is issued. The default is enabled.
8. Select which security protocol is active when security is enabled from the Active Protocol menu. Specifies the active authentication protocol for RMI/IIOP requests when security is enabled.

   WebSphere Application Server includes the Object Management Group (OMG) protocol called CSIv2, which supports increased vendor interoperability and additional features. If all servers in your entire security domain are Version 6 servers, it is best to specify CSIv2 as your protocol. The default is both CSIv2 and z/SAS.
9. Select which authentication mechanism is active which security is enabled from the Active Authentication Mechanism menu. The Active Authentication Mechanism menu specifies the authentication mechanism which is active when security is enabled. In WebSphere Application Server Version 6, Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA) are the supported authentication mechanisms. Only LTPA is configurable on WebSphere Application Server Network Deployment. SWAM is not configurable on WebSphere Application Server Network Deployment.
10. Use the Active user registry menu to specify the user registry that is active when security is enabled. You can configure settings for one of the following user registries:
    - Local operating system.

      The implementation is a SAF compliant registry such as the Resource Access Control Facility (RACF), which is shared in an MVS sysplex.
    - LDAP user registry. The LDAP User Registry settings are used when users and groups reside in an external LDAP directory. When security is enabled and any of these properties are changed, go to the Global Security panel and click **OK** or **Apply** to validate the changes.
    - Custom user registry.

    The default user registry is local OS. However, you can configure the supported user registries under the User registries section of this administrative console panel.
11. Click the **Use the Federal Information Processing Standard (FIPS)** option if you are using a FIPS-certified JSSE. WebSphere Application Server Version 6 supports a channel framework that uses IBMJSSE2. IBMJSSE2 uses IBMJCEFIPS for cryptographic support when you enable the **Use the Federal Information Processing Standard (FIPS)** option.
12. Click **OK**.

    This panel performs a final validation of the security configuration. When you click **OK** or **Apply** from this panel, the security validation routine is performed and any problems are reported at the top of the page. When you complete all of the fields, click **OK** or **Apply** to accept the selected settings. Click **Save** (at the top of the panel) to persist these settings out to a file. If you see any informational messages in red text color, then there is a problem with the security validation. Typically, the message indicates the problem. So, review your configuration to verify that the user registry settings are accurate and the correct registry is selected. In some cases, the LTPA configuration may not be fully specified. See “Global security settings” on page 886 for detailed information.

Configuration is successful when error messages do not display at the top of the panel.

Enabling global security on a base application server node:

Global security activates a number of WebSphere security settings. Most of the settings receive their default value from the installation scripts, run during server installation. The following is a checklist for enabling global security on a base application server node, using the SAF-based (LocalOS) user registry and LTPA authentication:

1. Verify that you are running W510200 or later.
2. Verify that **Configure for local OS security registry** is set to **Y** in the customization dialog security domain setup.

3. Verify that the customization dialog jobs BBOSBRAK and BBOCBRAK, which create keyrings and certificates, were run and completed successfully.
4. Start the server if it is not already up.
5. Access the administrative console. You can use any user ID. A password is not necessary.
6. Specify LTPA as the authentication mechanism.
   a. Click **Security> Global security**.
   b. Under Authentication, click **Authentication mechanisms > LTPA** .
   c. Enter a password and confirm the password by entering it again.
   d. Click **Apply** and **Save**.
7. Specify the SAF properties.
   a. Click **Security > Global security**.
   b. Under User registries, click **Local OS**.
   c. Under Additional properties, click **z/OS SAF properties**.

      If you set **Use SAF EJBROLE profiles to enforce J2EE roles** to **Y** in the customization dialog, then the Authorization option is selected and the correct EJBROLE profiles for initial security setup were created by the BBOSBRAK and BBOCBRAK jobs.

      If you need to use SAF authorization for Java 2 Platform, Enterprise Edition (J2EE) roles, and you did not set the **Use SAF EJBROLE profiles to enforce J2EE roles** option in the customization dialog, then you must create the EJBROLE profiles manually, select the Authorization option , and click **Apply** and **Save**. For more information, see “Controlling access to console users when using a Local OS Registry” on page 849.

      If you wish to use WebSphere Application Server authorization for J2EE roles, verify that the Authorization option is deselected. If you change the setting, click **Apply** and **Save**.
8. Set the EnableTrustedApplications property in the custom properties for global security.
   a. Click **Security > Global security**.
   b. Under Additional properties, click **Custom properties**.
   c. Verify that the EnableTrustedApplications property value is set to `true`. If the property value is `false`, click the property name and change the value to `true`.
   d. Click **Apply** and **Save**
9. Verify that the global security options are correct.
   a. Click **Security > Global security**.
   b. Verify that the **Enable global security** option is selected.
   c. Select the appropriate authentication mechanism.

      WebSphere Application Server includes the Object Management Group (OMG) protocol called CSIv2, which supports increased vendor interoperability and additional features. If all servers in your entire security domain are Version 6.0.x servers, it is best to specify CSIv2 as your protocol. The default is both CSIv2 and z/SAS.
   d. Under Active authentication mechanism, select **Lightweight Third Party Authentication (LTPA)**.
   e. Under Active user registry, select **Local OS (single, stand-alone server or sysplex and root administrator only**.
   f. Click **Apply** and **Save**.
10. Restart the server and connect to the administrative console using your browser. The server should successfully redirect you to the SSL port, where you might receive certificate warnings from your browser. Then, you should see the login page where you can enter the valid administrative user ID and password.

Disabling global security:

Complete the following steps to disable global security.

1. Log into the administrative console.
2. Click **Security > Global security**.
3. Deselect the **Enable global security** option.
4. Restart the server.

If global security is not working properly, it can cause the server to not start, or start without providing you with the ability to log on. To disable global security in this case, edit the server `security.xml` file. The `security.xml` file can be found in the *mount_point*`/AppServer/config/cells//AppServer/config/cells/`*cell_name*`/` directory.

To disable global security, edit the `security.xml`. Search for the line that begins with the following tag: `<security:Security:`. In that line search for **Enabled**. The word following **Enabled** is **True**. Change it to **False**. **Save** the file. Restart the server. Global security is now disabled.

Enabling global security on a base application server node:

Global security activates a number of WebSphere security settings. You may not understand all of these settings or know what value they should be set to. Fortunately, most of the settings receive their default value from the installation scripts, run during server installation. The following is a checklist for enabling global security on a Base Application Server Node:

1. Ensure that you are running W500101 or later.
2. Ensure that the installation scripts were run, including the security panel. On the security panel, make sure you selected the option **generate RACF commands** for the above.
3. Ensure that you ran the job that submits the RACF commands created by the installation scripts. This builds the keyrings and certificates.
4. Start the server if it is not already up.
5. Go to the admin console. Sign in using any user ID. A password is not needed.
6. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**. Fill in a password and confirm it by entering it again. Click **Apply** and **Save**.
7. Click **Security > Global security**. Under User registries, click **Local OS**. Under additional properties, click **Custom Properties**. If you want WebSphere to use RACF EJBROLE profiles for determining if a user has a role, select

   `com.ibm.security.SAF.authorization`

   and

   `com.ibm.security.SAF.delegation`

   and set them to true. Otherwise, leave them set to false. If you change them, click **Apply** and **Save**. If you chose to use EJBROLE profiles, use RACF to PERMIT your administrative user IDs to the EJBROLE class profile *administrator*. If you chose not to use EJBROLE profiles, you should click **System Administration > Console Users**, and add your user IDs as administrators. Click **Apply** and **Save**.
8. Click **Security > Global security**. Under User registries, click **Local OS**. Under Additional properties, click **Custom properties**.
9. Click **EnableTrustedApplications** and set its value to *true*. Click **Apply** and **Save**.
10. Click **Security > Global Security**. Select the **Enable global security** option and then deselect the **Enforce Java 2 Security** option. The *Active Protocol* should be `CSI` and `SAS`. The *Active Authentication Mechanism* should be `LTPA`. The *Active User Registry* should be **Local OS**. Click **Apply** and **Save**.

Now you can cancel the server, restart it, and connect to the administrative console using your browser. The server should successfully redirect you to the SSL port, where you get the usual certificate warnings. Then you should see the login page where you can enter the valid administrative user ID and password.

Disabling global security:

You can disable global security through the administrative console. If global security is not working properly, it can cause the server to not start, or to start without providing you with the ability to log into the administrative console. If you can log into the administrative console, you can disable global security by completing the following steps:

1. Log into the administrative console and select **Security > Global Security**.

2. Deselect the **Enable global security** option.
3. Restart the server.

If you cannot log into the administrative console and you must disable global security, edit the server `security.xml` file. You can find the `security.xml` file, by default, in the *install_dir*/AppServer/profiles/*profile_name*/config/cells/*cell_name*/. To disable global security, edit the `security.xml` file using the following steps:

1. Search for the line that begins: `<security:Security:`
2. In that line, search for **Enabled**.
3. Change the **Enabled** value to **False**.
4. **Save** the file.
5. Restart the server.

Global security is disabled.

Selecting a user registry:

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.

WebSphere Application Server for z/OS is designed with the capability to support multiple operating systems or operating environment-based user registries (z/OS SAF registry) and most of the major Lightweight Directory Access Protocol (LDAP)-based user registries. You can use the custom LDAP feature to support any LDAP server by setting up the correct configuration (user and group filters). However, support is not extended to these custom LDAP servers because there are many possibilities that cannot be tested.

In addition to Local OS and LDAP registries, WebSphere Application Server also provides a plug-in to support any registry by using the custom user registry feature. The custom user registry feature allows the configuration of any user registry that is not made available through the security configuration panels of the WebSphere Application Server. The possibilities are endless with the implementation of the UserRegistry interface. This 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 OS or LDAP. In such a case, 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 user 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.

Before configuring the user registry, decide which registry to use. The choices of user registry include:
- Local OS - SAF-based
- LDAP
- Custom user registry

Though different types of user registries are supported, only a single user registry can be active at one time. All processes in WebSphere Application Server can use one active registry. Configuring the correct registry is a prerequisite to assigning users and groups to roles for applications. This is usually done as part of enabling global security. Restart the servers and assign users and groups to roles for all your applications.

Selecting an authentication mechanism:

Once you have your system up and running, the next step in setting up security is to select an authentication mechanism. An authentication mechanism defines rules about security information (for example, 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 valid 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 accounts 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 successfully authenticated client user. Not all credentials are created equal. The abilities of the credential are determined by the configured authentication mechanism.

Although this product provides several authentication mechanisms, only a single active authentication mechanism can be configured at once. The active authentication mechanism is selected when configuring WebSphere Application Server global security. WebSphere Application Server for z/OS supports the following authentication mechanisms:

- Simple WebSphere Authentication Mechanism (SWAM)
- Lightweight Third Party Authentication (LTPA)
- Integrated Cryptographic Service Facility (ICSF)

Authorization checking:

Each controller, servant, and client must be associated with an MVS user ID. When a request flows from a client to the server or from a server to another server, WebSphere Application Server for z/OS passes the user identity (client or server) with the request. This way, each request is performed on behalf of the user identity and the system checks to see if the user identity has the authority to make such a request.

There are three distinct levels of authorization checking.

1. Operating system-level security

   This first level of authentication is required by z/OS to protect its resources through the use of a System Authorization Facility (SAF) credential. This security is always enabled. For SAF, controllers, servants, and default clients must be associated with an MVS user ID. Operating system resources are accessible by applications when they are granted access to the MVS user ID of the servant.
2. Cell-level security

   The second level, which is in effect whenever WebSphere Application Server security is enabled at the cell level, is required to protect WebSphere's administrative resources.
3. Server security

   The third level, which is in effect whenever WebSphere Application Server security is enabled for a given server, is a set of authorization checking mechanisms required to control access to WebSphere J2EE applications. On a base server, the cell and server levels of security can be viewed as the same.

When WebSphere Application Server security is enabled, WebSphere administrative and Java 2 Platform, Enterprise Edition (J2EE) authorizations can be performed using the identity authenticated with the configured user registry.

When the user registry is configured to be `LocalOS`, the operating system and WebSphere identities are the same. If the Local OS user registry is active, or if pluggable identity mapping modules are in place to map WebSphere Application Server user identities to operating system (SAF) identities, authorization checking can be configured to use SAF EJBROLE profiles by setting the registry custom property `com.ibm.security.SAF.authorization` to **true**. Otherwise, WebSphere application bindings are used to provide user to role mappings.

Controlling access to console users when using a Local OS Registry:

The user registry and authorization settings for the cell control how you add console users. If the user registry custom property `com.ibm.security.SAF.authorization` is set to **true**, then System Authorization Facility (SAF) EJBROLE profiles are used to authorize console users. (For non-LocalOS user registries,

you must use identity mapping to map WebSphere identities to SAF user IDs). If `com.ibm.security.SAF.authorization` is set to **false**, the administrative console is used to authorize console users and groups.

Regardless of which type of registry or authorization setting is chosen, the configuration process authorizes the WebSphere configuration group (to which all WebSphere Server identities are permitted), and an MVS user ID for the WebSphere administrator identity to do the following tasks:

- Access all administrative console functions
- Use the administrative scripting tool when security is first enabled

When SAF authorization is selected on z/OS, the special subject of server is not used as the administrative user ID. (Note that the customization dialogs generate an administrative user, who is a member of the administrative group, which can be used for authorization.)

Using SAF Authorization to control access to Administrative functions: When SAF Authorization is selected during systems customization, administrative EJBROLE profiles for all administrative roles are defined by the RACF jobs generated using the Configuration Dialog. If SAF Authorization is selected subsequently, issue the following RACF commands (or equivalent security server commands) to enable your servers and administrator to administer WebSphere Application Server:

```
RDEFINE EJBROLE (optionalSecurityDomainName.)administrator UACC(NONE)
RDEFINE EJBROLE (optionalSecurityDomainName.)monitor       UACC(NONE)
RDEFINE EJBROLE (optionalSecurityDomainName.)configurator  UACC(NONE)
RDEFINE EJBROLE (optionalSecurityDomainName.)operator      UACC(NONE)

PERMIT (optionalSecurityDomainName.)administrator CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
PERMIT (optionalSecurityDomainName.)monitor       CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
PERMIT (optionalSecurityDomainName.)configurator  CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
PERMIT (optionalSecurityDomainName.)operator      CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
```

If additional users require access to administrative functions, you can permit a user to any of the above roles as follows by issuing the following RACF command:

```
PERMIT (optionalSecurityDomainName.)rolename   CLASS(EJBROLE)  ID(mvsid) ACCESS(READ)
```

You can give a user access to all administrative functions by connecting it to the configuration group:

```
CONNECT  mvsid  GROUP(configGroup)
```

Using WebSphere Authorization to control access to administrative functions: To assign users to administrative roles, go to the administrative console, expand **System Administration**, and click **Console Users** or **Console Groups**, and then add the user's WebSphere Application Server for z/OS user identities as desired. For more information on console user roles, refer "Administrative console and naming service authorization" on page 907.

Note:

- When SAF Authorization is in effect, WebSphere Application Server authorization, as specified in the administrative console, is ignored.
- SAF role names are case-sensitive.

Summary of controls: Each controller, servant, and client must have its own MVS user ID. When a request flows from a client to the cluster or from a cluster to a cluster, WebSphere Application Server for z/OS passes the user identity (client or cluster) with the request. Thus, each request is performed on behalf of the user identity and the system checks to see if the user identity has the authority to make such a request. The tables in this article outline System Authorization Facility (SAF) and non-SAF authorizations.

Summary of z/OS security controls independent of global security setting

In a WebSphere Application Server for z/OS configuration, there are many different types of processes:

- Deployment managers
- Node agents
- Location service daemons
- WebSphere Application Servers

Each of these can be viewed as either a WebSphere Application Server for z/OS controller process or pair of processes (a controller and servant).

Each controller and servant must run under a valid MVS user ID assigned as part of the definition of a started task. This MVS user ID must have a valid UNIX Systems Services user identity (UID) and be connected to WebSphere configuration group that is common to all servers in the cell with a valid MVS and UNIX System Services group identity (GID) identity.

The following table summarizes the controls used to grant authorizations needed by these controllers and servants to access operating system resources. By understanding and using these controls, you can control all resource accesses in WebSphere Application Server for z/OS.

Table 9. Summary of controls and SAF authorizations

Control	Authorization
DATASET class	Access to data sets
DSNR class	Access to Database 2 (DB2)
FACILITY class (BPX.WLMSERVER)	Access to the BPX.WLMSERVER profile to perform Workload Management (WLM) enclave management in the servant. Without this access, classification is not performed.
FACILITY class (IMSXCF.OTMACI)	Access to Open Transaction Manager Access (OTMA) for Information Management System (IMS), and access to the BPX.WLMSERVER profile
HFS file permissions	Access to Hierarchical File System (HFS) files
LOGSTRM class	Access to log streams
OPERCMDS class	Access to `startServer.sh` shell script and Integral JMSProvider
SERVER class	Access to controller by a servant
STARTED class	Associate user ID (and optionally group ID) to start procedure
SURROGAT class (*.DFHEXCI)	Access to EXCI for Customer Information Control System (CICS) access

The customization dialogs and Resource Access Control Facility (RACF) customization jobs set these up for the initial server settings for the `*'ed` profiles.

Note: Examples of authorizations for the other profiles can be found in the generated `exec` file in `HLQ.DATA(BBOWBRAC)`. The selection of an identity to be used for authorization to native connector resources (CICS, DB2, IMS) is dependent on the:

- Type of connector
- Resource authentication (resAuth) setting of the deployed application
- Availability of an alias

- Global security setting

Resource managers such as DB2, IMS, and CICS have implemented their own resource controls, which control the ability of clients to access resources. When resource controls are used by DB2, use the DSNR RACF class (if you have RACF support) or issue the relevant DB2 GRANT statements. You can:

- Access OTMA for IMS through the FACILITY Class (`IMSXCF.OTMACI`)
- Access EXCI for CICS through the SURROGAT class (`*.DFHEXCI`)
- Control access to data sets through the DATASET class and HFS files through file permission

Note that MVS SAF Authorization to all other MVS subsystem resources accessed by J2EE applications is typically performed using the identity of the servant MVS user ID. Refer to "Understanding Java 2 Platform, Enterprise Edition identity and an operating system thread identity" on page 896 for more information.

The BPX.WLMSERVER profile in the FACILITY class is used to authorize an address space to use the Language Environment (LE) run-time services that interface with workload management (WLM) to perform workload management within a server region. These LE run-time services are by used by WebSphere Application Server to extract classification information from enclaves and to manage the association of work with an Enclave. Because unauthorized interfaces are used to manipulate WLM enclaves for server region work that has not been passed from a controller to a servant, WebSphere Application Server servants should be permitted READ access to this profile. Without this permission, attempts to create, delete, join, or leave a WLM enclave fails with a `java.lang.SecurityException`.

Summary of z/OS security controls in effect when global security is enabled

When global security is enabled, SSL must be available for encryption and message protection. In addition, authentication and authorization of J2EE and administrative clients is enabled.

The FACILITY class authorization needed for SSL services and the definition of SAF keyrings are required when global security is enabled. The remainder of the z/OS security controls described here are valid only when LocalOS is chosen as the registry. For a description of non-z/OS-specific WebSphere Application Server controls, refer to Assembling secured applications, Deploying secured applications, and Managing security.

When a request flows from a client to the WebSphere Application Server or from a cluster to a cluster, WebSphere Application Server for z/OS passes the user identity (client or cluster) with the request. Thus each request is performed on behalf of the user identity and the system checks to see if the user identity has the authority to make such a request. The tables in this article outline z/OS specific authorizations using SAF.

The following table summarizes the controls used to grant authorizations to resources. By understanding and using these controls, you can control access to all resources in WebSphere Application Server for z/OS.

Table 10. Summary of controls and SAF authorizations

Control	Authorization
CBIND class	Access to a cluster
EJBROLE or GEJBROLE class	Access to methods in enterprise beans
FACILITY class (IRR.DIGTCERT.LIST and IRR.DIGTCERT.LISTRING)	SSL key rings, certificates, and mappings
FACILITY Class (IRR.RUSERMAP)	Kerberos credentials

Table 10. Summary of controls and SAF authorizations (continued)

Control	Authorization
PTKTDATA class	PassTicket enabling in the sysplex
Set OS Thread Identity to RunAs Identity	J2EE cluster property used to enable the execution identity for non-J2EE resources

Cluster authorizations:

This section discusses the kinds of authorization checking WebSphere Application Server for z/OS does for a clusters. Servants must have access to profiles in the RACF SERVER class. This controls whether a servant can call authorized routines in the controller.

The following explains the kinds of authorization checking WebSphere Application Server for z/OS does for clusters.

1. Servants must have access to profiles in the RACF SERVER class. This controls whether a servant can call authorized routines in the controller.

   Controllers do not require such access control. Only authorized programs, loaded from Authorized Program Facility (APF) libraries, run in controllers.
2. Resource managers such as DB2, IBM Information Management System (IMS), and Customer Information Control System (CICS) have implemented their own resource controls, which control the ability of applications to access resources.

   When resource controls are used by DB2, all controllers and servants need to be granted access to the relevant resources. You can grant access by using the DSNR RACF class (if you have RACF support) or by issuing the relevant DB2 GRANT statements.

   Access to OTMA for IMS access is accomplished through the FACILITY Class (IMSXCF.OTMACI). Access to EXCI for CICS is accomplished through the SURROGAT class (*.DFHEXCI).

   You can control access to data sets through the DATASET class and HFS files through file permissions.

Specifics about server process authorization checking: To control access to WebSphere Application Server for z/OS resources:

- As a general rule, give greater authority to controllers and less authority to servants.

Table 11. Level of trust and authority for regions

Region	Level of trust and access authority
Controller	**Note:** • Contains WebSphere Application Server for z/OS system code. • Trusted, runs APF-authorized • Contains communication ports and manipulation of SAF client identities
Servant	**Note:** • Contains WebSphere Application Server for z/OS system code, application code, and pluggable service providers (such as jdbc drivers) • Supports Java 2 Security to protect sensitive data and system services • Untrusted

- Regarding the WebSphere Application Server for z/OS run-time clusters, the general rule is to give less authority to the location service daemon, and greater authority to the node, as explained in the table below:

Table 12. Assigning authorities to WebSphere Application Server for z/OS run-time cluster control and servants

Run-time Cluster	Region	Required Authorities
Location service daemon	Control	• STARTED class • Access to WLM services • Access to DNS • OPERCMDS access to START, STOP, CANCEL, FORCE, and MODIFY other clusters • IRR.DIGTCERT.LIST and IRR.DIGCERT.LISTRING in FACILITY (SSL)
Node	Control	STARTED class
Controller	Control	• SSL • Kerberos • READ authority to the SERVER class, • OPERCMDS access to START, STOP, CANCEL, FORCE and MODIFY other servers
Servant	Control	The following classes: • OTMA • SERVER • DSNR, • DATASET • SURROGATE • STARTED • LOGSTREEAM

- Remember to protect the Resource Recovery Services (RRS) log streams. By default, UACC is READ.
- Protect the WebSphere Application Server for z/OS properties XML files, especially if they contain passwords. For more information, see the WebSphere Application Server variables in the administrative console or the documentation.
- Deployment Manager also needs permission to start and stop servers.

Using CBIND to control access to clusters:

You can use the CBIND class in RACF to restrict a client's ability to access clusters from Java Application Clients or other J2EE compliant servers. You will need READ permission to access clusters.

You can also use this class to specify which servers are trusted to assert identities (with no authenticator):

- z/OS Secure Authentication Services (z/SAS) identity assertion accepted
- Common Secure Interoperability Version 2 (CSIv2) identity assertion
- Web container HTTP transport

This validates an intermediate server to send certificates (`MutualAuthCBindCheck=true.certificates`). You can deactivate the class if you do not require this kind of access control.

Servers are either clustered or not clustered. The value of cluster_name is:

1. For a clustered server, the cluster_name used in these profiles is the cluster short name.
2. For an unclustered server, instead of a cluster_name a server custom property (ClusterTransitionName) is used.

Note: When you convert a server into a clustered server the ClusterTransitionName becomes the cluster's short name.

The following explains the CBIND authorization checking by WebSphere Application Server for z/OS.

1. You can use the CBIND class in RACF to restrict the ability of a client to access clusters, or you can deactivate the class if you do not require this kind of access control. There are two types of profiles WebSphere Application Server for z/OS uses in the CBIND class:

- One that controls whether a local or remote client can access clusters. The name of the profile has this form:

  `CB.BIND.cluster_name`

  where *cluster_name* is the name of the cluster.
- One that controls whether a client can invoke J2EE applications in a cluster. The name of the profile has this form:

  `CB.cluster_name`

  where *cluster_name* is the name of the cluster.

Note: When you add a new cluster, you must authorize all Java Client user IDs and Servers to have read access to the CB.**cluster_name** and CB.BIND.**cluster_name** RACF profiles.

Example: WSADMIN needs read authority to the CB.BBOC001 and CB.BIND.BBOC001 profiles:

```
PERMIT CB.BBOC001      CLASS(CBIND) ID(WSADMIN) ACCESS(READ)
PERMIT CB.BIND.BBOC001 CLASS(CBIND) ID(WSADMIN) ACCESS(READ)
```

2. You can also use the System Authorization Facility (SAF) CBIND class to indicate that a process is trusted to assert identities to WebSphere Application Server for z/OS. This usage is primarily intended for use by trusted intermediate servers who have already authenticated their callers.

   The intermediate server (or process) must establish its network identity to WebSphere Application Server for z/OS using SSL client certificates. This network identity is mapped to an MVS user ID by SAF security service. This mapped identity must be granted CONTROL access to the `CB.BIND.cluster_name` process in order to be authorized to assert identities.

   The use of CBIND profiles to establish trust is used by the following authentication mechanisms:
   - Web container HTTP transport (which validates unencrypted client certificates when the property: `MutualAuthCBindCheck=true` is set)
   - CSIv2 identity assertion for IIOP
   - z/SAS identity assertion accepted

   For example, WEBSERV needs to assert client certificates received from its callers: `PERMIT CB.BBOC001 CLASS(CBIND) ID(WEBSERV) ACCESS(CONTROL)`

Refer to “System Authorization Facility for role-based authorization” for more information.

System Authorization Facility for role-based authorization: **EJBROLE**: As an alternative to WebSphere authorization, Security Authorization Facility (SAF)-based authorization (for example, using the RACF EJBROLE profile) can be used to control a client’s access to Java 2 Platform, Enterprise Edition (J2EE) roles in EJB and Web applications, including the WebSphere administrative console application. If the user registry custom property `com.ibm.security.SAF.authorization` is set to **true**, then SAF EJBROLE profiles are used to authorize J2EE roles. (For non-LocalOS user registries, identity mapping must be in place to map WebSphere identities to SAF identities).

Defining EJBROLES belongs to the application deployment process. If the user ID has at least READ access to the EJBROLE profile defined in that corresponds to the J2EE role defined by the application, the user ID is considered to be *in Role*. (Do not be confused by the name EJBROLE. It is used for J2EE roles in both EJBs and Web applications.)

When an application deployer uses a role in the deployment descriptor of a component, the role name must be identical to the name of an EJBROLE profile. A security administrator defines EJBROLE profiles and permits SAF users or groups to the profiles. In order to be considered as eligible for a role, a user must have read access to the EJBROLE profile or must be connected to a SAF group that has read access.

The specification of a security domain prefix affects the specific EJBROLE profiles used by WebSphere Application Server for z/OS system resources when SAF authorization is chosen. When `SecurityDomainType = cellQualified`, the WebSphere Application Server for z/OS run time J2EE application EJBROLE profiles are done by the specification of a security domain prefix. This enables you to deploy the same application on different cells in the same sysplex, but have different user to role mappings if desired.

Example: Your application has two J2EE role names: `juniorTellers` and `seniorTellers`. These are mixed case roles.

In your SAF registry, you have an MVS group called `JTELLER` and `STELLE`R and a MVS user ID called BANKADM. The `JTELLER` group is required to access to the juniorTellers role, and the `STELLER` group is required to acces the seniorTellers role. The `BANKADM` user ID is required to access both roles.

You have two cells, both defined to use a security Domain prefix. The security domain names are `PRODCELL` and `TESTCELL`, respectively. The TEST1 user ID should have access to both roles, but only in the test environment `TESTCELL`.

If you wanted to deploy the same application in both cells, you must define distinct profiles using a RACF (or equivalent security subsystem) as follows.

If RACF is used as your security server, enable this by issuing the following commands:

```
/* the EJBROLE class must be active, this step is done by the customization dialogs  */
SETROPTS CLASSACT(EJBROLE)

/* first define the roles in RACF */
RDEFINE EJBROLE PRODCELL.juniorTellers UACC(NONE)
RDEFINE EJBROLE PRODCELL.seniorTellers UACC(NONE)

RDEFINE EJBROLE TESTCELL.juniorTellers UACC(NONE)
RDEFINE EJBROLE TESTCELL.seniorTellers UACC(NONE)

/* permit the appropriate users and groups to the various roles */
PERMIT PRODCELL.juniorTellers CLASS(EJBROLE)  ID(JTELLER BANKADM) ACCESS(READ)
PERMIT PRODCELL.seniorTellers CLASS(EJBROLE)  ID(STELLER BANKADM) ACCESS(READ)

PERMIT TESTCELL.juniorTellers CLASS(EJBROLE)  ID(TEST1) ACCESS(READ)
PERMIT TESTCELL.seniorTellers CLASS(EJBROLE)  ID(TEST1) ACCESS(READ)

/* refresh the EJBROLE class in RACF *
SETROPTS RACLIST(EJBROLE) REFRESH"
```

Grouping EJBROLES (GEJBROLE): The SAF interface also supports a grouping class for the EJBROLE class. This grouping class is called GEJBROLE. It is particularly useful when you have a need to give access to the same users or groups for several roles.

The GEJBROLE grouping class provides a capability not natively available in other J2EE servers. Using the J2EE security model, if we have several components or applications that use different role names for similar functions (such as Hire, Promote, GrantPayraise for managerial functions), there are several options to handle this issue:

- Adjust the applications' deployment descriptors so that they conform to the roles already defined in our enterprise (such as Managers). This is time consuming and error prone, especially since it might require a readjustment of the deployment descriptor each time the application was changed or reinstalled.
- Define the EJBROLE profiles for each of the roles required by the application. Then the users and groups to be given access to these roles would have to be permitted. This could become an administrative headache, since the same users and groups would be permitted to several different profiles with similar meanings.

- Use the grouping class to avoid the worst pitfalls of the other two options. You must still define EJBROLE profiles for each of the roles required by the application. Instead of permitting all of the same users and groups to the new profiles, create a profile (such as Supervisors) in the grouping class and add all of the new EJBROLE profiles to it. Every user and group that needs access to these roles can now be permitted in one place--the Supervisors profile. You can further avoid administrative work by simply adding our existing EJBROLE profile (Managers) to the grouping class profile (Supervisors).

This following explains the relation between GEJBROLES, EJBROLES and EJBROLES within the GEJBROLE (ADDMEM).

Tip: Implementing GEJBROLES includes:

1. Plan organizational role profiles in RACF class GEJBROLEs.
2. Create the access list by permitting user groups to the GEJBROLE profiles, then add roles to the GEJBROLE profiles.
3. A GEJBROLE with only one EJBROLE is OK.
4. Do not use a mixture of EJBROLE and GEJBROLE for permitting users to roles.
5. If possible, permit users to GEJBROLE profiles only.
6. Generally use GEJBROLE in preference to EJBROLE.

Enabling global security: Global 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 global security is activated. The settings include the authentication of users, the use of Secure Sockets Layer (SSL), the choice of user registry and Java 2 security. In particular, application security, including authentication and role-based authorization, is not enforced unless global security is active. Global security is disabled by default to simplify the installation of the server. However, after you build a server and install the administrative console, any user can log on to the administrative console and a password is not required. Global security is necessary to secure the administrative console. However, proper planning is required because incorrectly enabling global security can lock you out of the administrative console, or cause the server to abend.

Why turn on global security?

Turning on global security activates the settings that protect your server from unauthorized users. There might be some environments where no security is needed such as a development system. On these systems you can elect not to enable global security. However, in most environments you should keep unauthorized users from accessing the administrative console and your business applications. Global security must be enabled to restrict access.

What does global security protect?

The settings that are activated when global security is enabled include:
- 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)
- CBIND checks

Setting up Secure Sockets Layer security for WebSphere Application Server for z/OS:

This topic assumes you understand the SSL protocol and how cryptographic services system SSL works on z/OS or OS/390. Secure sockets layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy. Such components include the built-in HTTP transport, the

ORB (client and server), and the secure Lightweight Directory Access Protocol (LDAP) client. Configuring SSL is different between client and server with WebSphere Application Server. If you want the added security of protected communications and user authentication in a network, you can use secure sockets layer (SSL) security.

SSL is an integral part of the security provided by WebSphere Application Server for z/OS. It is activated when global security is enabled. When global security is enabled, SSL is always used by the administrative subsystem to secure administrative commands, the administration console, and communications between WebSphere Application Server processes.

The WebSphere Application Server for z/OS run time can optionally use SSL when server security is enabled in these cases:

- SSL is used to protect Web application when confidentiality is specified as a Web Application Security Constraint. A transport guarantee of CONFIDENTIAL or INTEGRAL guarantees that the communication between the Web client and the Web server is secured and is transported over HTTPS (HTTP SSL). In addition, you can use SSL to perform client authentication when the security constraint (CLIENT_CERT) is specified during application deployment .
- SSL can be used to protect Inter-ORB Protocol (IIOP) requests when SSL/TLS is supported (or required) in the Common Secure Interoperability version 2 (CSIv2) transport settings. These are set by clicking **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound transport** or **CSIv2 outbound transport**.
- SSL can be used to protect IIOP requests when z/OS Secure Authentication Services (z/SAS) protocols are selected. SSL is used with SSL basic authentication, SSL client authentication, z/SAS identity assertion, and z/SAS Kerberos. SSL client authentication and z/SAS identity assertion also uses SSL transmitted digital certificates to authenticate the sender of the request.
- SSL can be used to protect communications between an LDAP client and server when the active user registry is LDAP.

When configuring SSL, there are two types of SSL repertoires on WebSphere Application Server for z/OS. The type of repertoire relates to the underlying services used to process SSL.

- System SSL (SSSL repertoires) are required for Web container (HTTP Transports) SSL, and Inter-ORB (IIOP) SSL processing, both CSIv2 and zSAS SSL Transports. In addition a System SSL repertoire must be specified if the RMI connector is chosen for administrative requests. System SSL repertoires use a System Authorization Facility (SAF) Keyring to retrieve the personal certificate and trust stores of the Application Server. All system SSL repertoires for a given process must use the same SAF Keyring.
- Java Secure Socket Extension (JSSE) must be selected as the SSL repertoire type for administrative requests using the HTTP/SOAP Connector. JSSE repertoires can (with APAR PQ77586 applied) specify either a SAF keyring for the keystore or truststore, or an HFS file.

This topic gives a brief explanation of the SSL protocol and how SSL works on z/OS or OS/390. For information about the SSL protocol, go to the following Web site: http://home.netscape.com/eng/ssl3/ssl-toc.html

For more information about Cryptographic Services System SSL, go to the following Web site: z/OS System Secure Sockets Layer Programming.

Secure Sockets Layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP Transport, the ORB (client and server), and the secure LDAP client. Configuring SSL is different between client and server with WebSphere Application Server. If you want the added security of protected communications and user authentication in a network, you can use Secure Sockets Layer (SSL) security. The SSL support in WebSphere Application Server for z/OS has several objectives:

- To provide ways accepted by the industry to protect the security of messages as they flow across the network. This is often called *transport layer security*. Transport layer security is a function that provides

privacy and data integrity between two communicating applications. The protection occurs in a layer of software on top of the base transport protocol (for example, on top of TCP/IP).

SSL provides security over the communications link through encryption technology, ensuring the integrity of messages in a network. Because communications are encrypted between two parties, a third party cannot tamper with messages. SSL also provides confidentiality (ensuring the message content cannot be read), replay detection, and out-of-sequence detection.

- To provide a secure communications medium through which various authentication protocols can operate. A single SSL session can carry multiple authentication protocols, that is, methods to prove the identities of the parties communicating.

  SSL support always provides a mechanism by which the server proves its identity. The SSL support on WebSphere Application Server for z/OS allows these ways for the client to prove its identity:
  - Basic authentication (also known as SSL Type 1 authentication), in which a client proves its identity to the server by passing a user identity and password known by the target server.

    With SSL basic authentication:
    - A z/OS or OS/390 client can communicate securely with WebSphere Application Server for z/OS with a user ID and password as defined by the CSIv2 user name and password mechanism (GSSUP).
    - A WebSphere Application Server client can communicate securely with a WebSphere Application Server for z/OS server by using a MVS user ID and password.
    - Because a password is always required on a request, only simple client-to-server connections can be made. That is, the server cannot send a client's user ID to another server for a response to a request.
  - Client certificate support, in which both the server and client supply digital certificates to prove their identities to each other.

    When digital certificates are provided for authentication to WebSphere Application Server for z/OS the decrypted certificate is mapped to a valid user identity in the active user registry. Web applications can have thousands of clients, which makes managing client authentication an administrative burden. When Local OS is the active user registry on WebSphere Application Server for z/OS, SAF certificate name filtering allows you to map client certificates, without storing them, to MVS user IDs. Through *certificate name filtering*, you can authorize sets of users to access servers without the administrative overhead of creating MVS user IDs and managing client certificates for every user.
  - SSL support always provides a mechanism by which the server proves its identity. A variety of mechanisms can be used to prove the clients identity. The SSL v3 (and TLS) protocol provides for the ability for client digital certificates to optionally be exchanged. These certificates can be used for authentication.
  - CSIv2 identity assertion, which provides support for z/OS and OS/390 principals, X501 distinguished names, and X509 digital certificates.
  - Identity assertion, or trusted association, in which an intermediate server can send the identities of its clients to a target server in a secure yet efficient manner. This support uses client certificates to establish the intermediate server as the owner of an SSL session. Through the Resource Access Control Facility (RACF), the system can check that the intermediate server can be trusted (to confer this level of trust, CBIND authorization is granted by administrators to RACF IDs that run secure system code exclusively). Once trust in this intermediate server is established, client identities (MVS user IDs) need not be separately verified by the target server; those client identities are simply asserted without requiring authentication.
- To be securely interoperable with other products, such as:
  - CICS Transaction Server for z/OS
  - Other WebSphere Application Server versions
  - CORBA-compliant object request brokers

SSL is disabled by default and SSL support is optional. If you are running WebSphere Application Server for z/OS with security turned on, SSL is required by the administrative console.

If you choose to use SSL, there are two types of SSL repertoires from which you must choose:

- System SSL (SSSL) is the SSL repertoire type used for Web container and ORB transport.
- Java Secure Socket Extension (JSSE) is the SSL repertoire type used for the JMX SOAP Connector

The following table describes how an SSL connection works:

Stage	Description
Negotiation	After the client locates the server, the client and server negotiate the type of security for communications. If SSL is to be used, the client is told to connect to a special SSL port.
Handshake	The client connects to the SSL port and the SSL handshake occurs. If successful, encrypted communication starts. The client authenticates the server by inspecting the server's digital certificate. If client certificates are used during the handshake, the server authenticates the client by inspecting the client's digital certificate.
Ongoing communication	During the SSL handshake, the client and server negotiate a cipher spec to be used to encrypt communications.
First client request	The determination of client identity depends upon the client authentication mechanism chosen, which is one of the following: • CSIv2 user ID and password (GSSUP) • CSIv2 asserted identity • zSAS Kerberos • zSAS basic authentication assertediIdentities • zSAS asserted identities • CSIv2 client certificates • zSAS client certificates

Rules:

- Only server controllers and z/OS or OS/390 clients require access to Cryptographic Services System SSL. Your controllers and z/OS or OS/390 clients require access to the *hlq*. SGSKLOAD data set. Place SGSKLOAD into LPA. You must use system SSL to establish secure communications. It is recommended that the system SSL load library exist in the linklist and be under program control. Verify that the load library exists in the link list. To turn on program control for the library, issue the following RACF commands from a user ID that has the proper authority:

  ```
  RALTER PROGRAM * ADDMEM('hlq.SGSKLOAD' //NOPADCHK) UACC(READ)
  SETROPTS WHEN(PROGRAM) REFRESH
  ```

  For more information, see *z/OS System Secure Sockets Layer Programming*.
- Either a Java or C++ client on z/OS or OS/390 is interoperable with a WebSphere Application Server for z/OS or workstation Application Server, and can use SSL. CSIv2 security only supports Java clients on z/OS or OS/390.
- Part of the handshake is to negotiate the cryptographic specs used by SSL for message protection. There are two factors that determine the cipher specs and key sizes used:
  - The security level of the cryptographic services installed on the system, which determines the cipher specs and key sizes available to WebSphere Application Server for z/OS.
  - The configuration of the server through the administrative console allows you to specify SSL cipher suites.

  For more information, see *z/OS System Secure Sockets Layer Programming*.
- For z/OS system SSL sockets you must use RACF or an equivalent to store digital certificates and keys. Placing digital certificates and keys into a key database in the HFS is not an option.

Tip: To define SSL basic authentication security, you must first request a signed certificate for your server and a certificate authority (CA) certificate from the certificate authority that signed your server certificate. After you have received a signed certificate for your server and a CA certificate from the

certificate authority, you must use RACF to authorize the use of digital certificates, store server certificates, and server key rings in RACF, create an SSL repertoire alias, and define SSL security properties for your server through the administrative console.

For clients, you must create a key ring and attach to it the CA certificate from the certificate authority that issued the server's certificate. For a z/OS or OS/390 client, you must use RACF to create a client key ring and to attach the CA certificate to that key ring. For the client to authenticate the server, the server (actually, the controller user ID) must possess a signed certificate created by a certificate authority. The server passes the signed certificate to prove its identity to the client. The client must possess the CA certificate from the same certificate authority that issued the server's certificate. The client uses the CA certificate to verify that the server's certificate is authentic. Once verified, the client can be sure that messages are truly coming from that server, not someone else. For the server to authenticate the client, note that there is no client certificate that the client passes to prove its identity to the server. In the SSL basic authentication scheme, the server authenticates the client by challenging the client for a user ID and password.

See "Setting up a keyring for use by Daemon Secure Sockets Layer" on page 862 for information on creating a keyring for the daemon's MVS user ID.

SSL repertoires:

The Secure Sockets Layer (SSL) configuration repertoire allows administrators to define any number of SSL settings which can be used to make HTTPS, IIOPS or LDAPS connections.

Using the SSL configuration repertoire, you can pick one of the SSL settings defined here from any location within the administrative console which allows SSL connections. This simplifies the SSL configuration process since you can reuse many of these SSL configurations by simply specifying the alias in multiple places. The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and enterprise beans containers. Before deleting SSL configurations from the repertoire, remember that if an SSL configuration alias is referenced somewhere, and it is deleted here, an SSL connection will fail if the deleted alias is accessed.

Note: You can also create an alias, but first you must create an SSL configuration repertoire alias or entry. You can then select the alias later when a component is configured for SSL support.

If you choose to use SSL, there are two types of SSL repertoires from which you must choose:
- System SSL (SSSL) is the SSL repertoire type used for Web container and the object request broker (ORB) transport
- Java Secure Socket Extension (JSSE) is the SSL repertoire type used for the Java Management Extensions (JMX) Simple Object Access Protocol (SOAP) Connector

Defining Secure Sockets Layer security for servers:

You need to request a certificate authority (CA) certificate and a signed certificate for your server. If you plan to implement Secure Sockets Layer (SSL) client certificate support, you must also have certificate authority certificates from each certificate authority that verifies your client certificates. You must have a user ID with the authority to use the RACDCERT command in the Resource Access Control Facility (RACF) (for example, SPECIAL authority).

Complete the following steps for RACF to authorize the server to use digital certificates. SSL uses digital certificates and public and private keys. If your application server uses SSL, you must use RACF to store digital certificates, and you must use public and private keys for the user identities under which the server controllers run.

1. For each server that uses SSL, create a key ring for the controller user ID of that server. **Example**: Your controller is associated with the user ID called ASCR1. Issue the following command:

   ```
   RACDCERT ADDRING(ACRRING) ID(ASCR1)
   ```

2. Receive the certificate for your application server from the certificate authority. **Example**: You requested a certificate and the certificate authority returned the signed certificate to you, which you stored in a file called ASCR1.CA. Issue the following command:

   ```
   RACDCERT ID (ASCR1) ADD('ASCR1.CA') WITHLABEL('ACRCERT') PASSWORD('password')
   ```

3. Connect the signed certificate to the controller user ID's key ring and make the certificate the default certificate. **Example**: Connect the certificate labeled ACRCERT to the key ring ACRRING owned by ASCR1. Issue the following command:

   ```
   RACDCERT ID(ASCR1) CONNECT (ID(ASCR1) LABEL('ACRCERT') RING(ACRRING) DEFAULT)
   ```

4. If you plan to have the server authenticate clients (SSL client certificate support), complete the following steps:
   a. Receive each certificate authority (CA) certificate that verifies your client certificates. **Example**: Receive the CA certificate that will verify a client with user ID CLIENT1. That certificate is in a file called USER.CLIENT1.CA. Issue the following command:

      ```
      RACDCERT ADD('USER.CLIENT1.CA') WITHLABEL('CLIENT1 CA') CERTAUTH
      ```

   b. Give each CA certificate the CERTAUTH attribute.

      Connect each client's certificate authority (CA) certificate to the controller user ID's key ring.

      Example: Connect the CLIENT1 CA certificate to the ring ACRRING owned by ASCR1.

      ```
      RACDCERT ID(ASCR1) CONNECT(CERTAUTH LABEL('CLIENT1 CA') RING(ACRRING))
      ```

5. Give read access for IRR.DIGTCERT.LIST and IRR.DIGTCERT.LISTRING in the RACF FACILITY class to the controller user ID. **Example**: Your controller user ID is ASCR1. Issue:

   ```
   PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(ASCR1) ACC(READ)
   PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(ASCR1) ACC(READ)
   ```

You are done with the RACF phase when the RACF commands succeed.

Setting up a keyring for use by Daemon Secure Sockets Layer:

Modify the customization job commands generated in BBOCBRAK (or HLQ.DATA(BBODBRAK) on WebSphere Application Server Network Deployment) to perform these steps:

1. Create a keyring for the daemon's MVS user ID to own. Generally, this is the same keyring name that was created for your application servers. Issue the following TSO command: `RACDCERT ADDRING(keyringname) ID(daemonUserid)`
2. Generate a digital certificate for the daemon's MVS user ID to own. Issue the following TSO command: `RACDCERT ID (daemonUserid) GENCERT SUBJECTSDN(CN('create a unique CN') O('IBM')) WITHLABEL('labelName') SIGNWITH(CERTAUTH LABEL('WebSphereCA'))`
3. Connect the generated certificate to the daemon's keyring. Issue the following TSO command: `RACDCERT ID(daemonUserid) CONNECT (LABEL('labelName') RING(keyringname) DEFAULT)`
4. Connect the certificate authority (CA) certificate to the server's keyring. Issue the following TSO command: `RACDCERT CONNECT (CERTAUTH LABEL(WebSphereCA) RING(keyringname))`

Tip: The CA certificate that is generated during configuration (WAS Test CertAuth) is an example. Use the CA you normally use to create user certificates, and connect the CA certificate to the daemon and server keyrings.

Setting up a keyring for use by cryptographic services:

Modify the customization job commands generated in BBOCBRAK (or HLQ.DATA(BBODBRAK) on WebSphere Application Server Network Deployment) to perform these steps:

1. Create a keyring for the daemon's MVS user ID to own. Generally, this is the same keyring name that was created for your application servers. Issue the following TSO command: `RACDCERT ADDRING(keyringname) ID(daemonUserid) ICSF`

2. Generate a digital certificate for the daemon's MVS user ID to own. Issue the following TSO command:
   `RACDCERT ID (daemonUserid) GENCERT SUBJECTSDN(CN('create a unique CN') O('IBM'))
   WITHLABEL('labelName') SIGNWITH(CERTAUTH LABEL('WebSphereCA')) ICSF`
3. Connect the generated certificate to the daemon's keyring. Issue the following TSO command:
   `RACDCERT ID(daemonUserid) CONNECT (LABEL('labelName') RING(keyringname) DEFAULT) ICSF`
4. Connect the certificate authority (CA) certificate to the server's keyring. Issue the following TSO command: `RACDCERT CONNECT (CERTAUTH LABEL(WebSphereCA) RING(keyringname)) ICSF`

Tip: The certificate authority (CA) certificate that is generated during configuration (WAS Test CertAuth) is an example. Use the CA you normally use to create user certificates, and connect the CA certificate to the daemon and server keyrings.

Defining SSL security for clients and servers:

Note that this assumes you use z/OS Security Server (RACF) as your security server. You must obtain a copy of the certificate authority (CA) certificate used to sign the server certificates. The server certificates connect your client to the server. You must also have a user ID with the appropriate authority (such as SPECIAL) to use the z/OS Security Server Resource Access Control Facility (RACF) RACDCERT command. For more information on the RACDCERT command, refer to z/OS Security Server RACF Command Language Reference (SA22-7687-05), available at http://www.ibm.com/servers/eserver/zseries/zos/bkserv/r5pdf/secserv.html. For more information on the RACF in general, refer to z/OS Security Server RACF Security Administrator's Guide (SA22-7683-05), available at http://www.ibm.com/servers/eserver/zseries/zos/bkserv/r5pdf/secserv.html.

Complete the following RACF steps to allow the client to use digital certificates. Simple Object Access Protocol (SOAP), Secure Socket Layer (SSL), and Java Secure Socket Extensions (JSSE) use digital certificates that have public and private keys. If your client uses SOAP, SSL or JSSE, you must use RACF to store digital certificates that have public and private keys for the user identities under which the client runs.

1. For each administrative client program that uses SOAP, create a keyring for the client user ID. For example, if your client is running with a user ID called CLIENTID, issue the following command:
   `RACDCERT ADDRING(ACRRING) ID(CLIENTID)`
2. The keyring created in the step above must include the public certificate of any certificate authority (CA) certificates that are required to establish trust in the servers to which your administrative client connects to. For each CA certificate complete the following steps:
   a. Determine whether this CA certificate is currently stored in RACF. If so, record the existing certificate label. If not you must:
      1) Receive each CA certificate used to sign a server certificate. For example, to receive the CA certificate that is stored in the `USER.SERVER1.CA` file and that verifies a server with the user ID SERVER1, issue the following command:
         `RACDCERT ADD('USER.SERVER1.CA') WITHLABEL('SERVER1 CA') CERTAUTH`
      2) Connect each server's CA certificate to the client user ID's keyring. For example, to connect the SERVER1 CA certificate to the ring ACRRING owned by CLIENTID:
         `RACDCERT ID(CLIENTID) CONNECT(CERTAUTH LABEL('SERVER1 CA') RING(ACRRING))`
3. If the servers your administrative client connect to implements SSL client certificate support, you must create certificates for your client and add them to the server keyrings. Refer to Defining SSL security for servers for instructions on setting up keyrings for the servers.
4. Give READ access for the IRR.DIGTCERT.LIST and IRR.DIGTCERT.LISTRING profiles in the RACF FACILITY class to the client user ID. For example, if your client user ID is CLIENTID, issue the following command:
   ```
   PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(CLIENTID) ACC(READ)
   PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CLIENTID) ACC(READ)
   ```

You are done with the RACF phase when the RACF commands have run successfully.

Steps to create a new System SSL repertoire alias:

You must start the administrative console.

The steps outline the necessary actions to generate a new System Secure Sockets Layer (SSL) repertoire alias. Using the SSL configuration repertoire, you can pick one of the SSL settings defined here from any location within the administrative console that allows SSL connections. This simplifies the SSL configuration process since you can reuse many of these SSL configurations by simply specifying the alias in multiple places.

1. Click **Security > SSL** on the left-hand navigation tree to open the SSL Configuration Repertoires panel.
2. To create a new System SSL alias, select the check box next to the word **Alias** and click on the **New SSSL repertoire** button near the top of the panel. The System SSL Repertoire panel appears.
3. Enter the alias name in the **Alias** field.
4. Specify the SSL Resource Access Control Facility (RACF) key ring in the **Key file name** field. All repertoires used by the same server (such as HTTPS, CSIV2, z/SAS) must have the same keyring name. If the keyring names are not the same, the HTTPS keyring name is used to initialize the server. If you specify the wrong RACF key ring, the server gets an error message at run time.
5. **Optional:** Select the **Client authentication** option for your authentication protocol. This option enables client authentication to occur if this repertoire is selected for HTTPS. However, the value is ignored if you use using Common Secure Interoperability Version 2 (CSIv2) or z/OS Secure Authentication Services (z/SAS).

   To enable client authentication for CSIv2, click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**. Select the appropriate option for **Client certificate authentication**.

   To enable client authentication for z/SAS, click **Security > Global security**. Under Authentication, click **Authentication protocol > z/SAS authentication**. Select the **Client certificate** option.
6. Select *High*, *Medium*, or *Low* from the **Security level** menu to specify the high, medium, or low set of cipher suites. If you add specific cipher suites on this panel, those cipher suites take precedence over the high, medium, or low specification. If a cipher list is specified, WebSphere Application Server uses the list. If the cipher list is empty, WebSphere Application Server uses the high, medium, low specification. The following list explains these specifications:

   High 128-bit cipher suites with digital signature.

   Medium
   40-bit cipher suites with digital signature.

   Low No encryption is used, but digital signature is used.
7. Specify the SSL V3 timeout value in the **V3 timeout** field. This value is the length of time, in seconds, that the system holds session keys. The range is 0-86400 (1 day). The default is 600 seconds.
8. Select the cipher suites that you want to add from the **Cipher suites** menu. By default, this is not set, and the cipher suites available are determined by the value of the Security Level (*High*, *Medium*, or *Low*). A cipher suite is a combination of cryptographic algorithms used for an SSL connection.
9. Click **OK** when you have made all your selections.

Creating a new Java Secure Socket Extension repertoire alias:

The following steps describe how to generate a new Java Secure Socket Extension (JSSE) repertoire alias. Using the JSSE repertoire, you can pick one of the JSSE repertoire settings defined here from any location within the administrative console. This simplifies the JSSE repertoire configuration process because you can reuse many of these JSSE configurations by simply specifying the alias in multiple places.

1. Click **Security > SSL** on the left-hand navigation tree to open the SSL Configuration Repertoires panel.
2. To create a new JSSE repertoire, click **New JSSE repertoire** near the top of the panel. The JSSE Repertoire panel appears.
3. Enter the alias name in the **Alias** field.
4. **Optional:** Select the **Client authentication** option for your authentication protocol. This option enables client authentication to occur if this repertoire is selected for HTTPS. However, the value is ignored if you use using Common Secure Interoperability Version 2 (CSIv2) or z/OS Secure Authentication Services (z/SAS).

   To enable client authentication for CSIv2, click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**. Select the appropriate option for **Client certificate authentication**.

   To enable client authentication for z/SAS, click **Security > Global security**. Under Authentication, click **Authentication protocol > z/SAS authentication**. Select the **Client certificate** option.
5. Select *High*, *Medium*, or *Low* from the **Security level** menu to specify the high, medium, or low set of cipher suites. If you add specific cipher suites on this panel, those cipher suites take precedence over the high, medium, or low specification. If a cipher list is specified, WebSphere Application Server uses the list. If the cipher list is empty, WebSphere Application Server uses the high, medium, low specification. The following list is an explanation of the high, medium, and low specifications:

   High 128-bit cipher suites with digital signature.

   Medium
   40-bit cipher suites with digital signature.

   Low No encryption is used, but digital signature is used.
6. Select the cipher suites that you want to add from the **Cipher suites** menu. By default, this is not set. The set of cipher suites available is determined by the value of the Security Level (*High*, *Medium*, or *Low*). A cipher suite is a combination of cryptographic algorithms used for an SSL connection.
7. Select the **Cryptographic token** option if hardware or software cryptographic support is available.
8. Indicate which JSSE provider that you are using by selecting either **Predefined JSSE provider** or **Custom JSSE provider** in the **Provider** field. WebSphere Application Server comes with the IBMJSSE provider predefined.

   If you are not using the IBMJSSE provider, configure a custom provider by selecting **Custom JSSE provider**. Under Additional properties, click **Custom Properties > New**. After specifying the custom provider, return to the JSSE repertoire panel.
9. Select an SSL or TLS protocol version.

   Note: The protocol chosen for the server must match the protocol chosen for the client. Also, in order for two servers to interoperate, they must use the same protocol.
10. Specify the name of the key file in the **Key file name** field. Specify the fully qualified path to the Secure Sockets Layer (SSL) key file that contains public keys and private keys. Type `safkeyring:///` if you are using a RACF key ring for the key file.
11. Specify the password needed to access the key file in the **Key file password** field. Type `password` if you are using a RACF key ring for the key store.
12. Select the format of the key file from the **Key file format** menu.
13. Click **OK** when you have made all your selections.

Daemon Secure Sockets Layer:

Use the administrative console panel to modify the port and Secure Sockets Layer (SSL) port settings and to specify the SSL settings (the SSL repertoire). The default repertoire is the same one used for the server, which is a SystemSSL IIOP repertoire. During daemon initialization the SSL usage initialization is

attempted if security is enabled and a valid repertoire is found. In order to turn *off* the daemon SSL port a cell-level WebSphere variable (`DAEMON_security_disable_daemon_ssl`) must be created and set to true. The default for this variable is *false*.

SSL can be used to protect locations in the SSL daemon using the Location Service Daemon if:

- Global security is enabled
- A daemon SSL repertoire is configured in the administrative console (the daemon SSL repertoire refers to a valid RACF keyring that is owned by the MVS user ID associated with the daemon process)
- A certificate and keyring have been defined

On the administrative console, click **System administration > Node groups >***sysplex_node_group_name*. Under Additional properties, click **z/OS location service**.

Location service daemon

This panel specifies the configuration settings for the location service daemon for this cell. Changes made to these settings to the entire cell and to the location service daemon instance on each node in the cell.

Job Name	BBODMNC	Specifies z/OS jobname of location service daemon.
Host Name	BOSSXXXX.PLEX1.L2.IBM.COM	Specifies host name to be used when contacting location service daemon.
Port	5755	Specifies port location service daemon listens on for unencrypted communication.
SSL Port	5756	Specifies port location service daemon listens on for encrypted communication.
SSL Setting	PLEX1Manager/DefaultIIOPSSL	Specifies a list of predefined SSL settings to choose from for connections. These are configured at the SSL repertoire panel.

You can use the customization dialog to specify authentication information, including the daemon's user ID, UID, and SSL port. This panel is located under **Server Customization**. RACF commands are generated to create a keyring for server use (the default is WASKeyring). The customization dialog generates the daemon keyring and the certificate. To generate the daemon keyring and certificate from the customization dialog, select **Security Domain > SSL Customization > Enable SSL on the Location Service Daemon.** If you type `Y` next to this option, the RACF commands are generated to do the following tasks:

- Create a daemon keyring and certificate
- Connect the certificate and certificate authority (CA) certificates to the keyring.

Important: This option does not control the use of the daemon SSL.

This is appropriate if the user IDs are the same, but if the daemon has a separate user ID, see Setting up a Keyring for use by WebSphere Application Server for z/OS. The values selected are picked up by the administrative console.

If the daemon process is assigned the same MVS user ID assigned to a secure WebSphere Application Server, the keyring you use to secure WebSphere Application Server can also be used to secure daemon requests. If the daemon process is not assigned the same MVS user ID assigned to a secure WebSphere Application Server, it is recommended that you perform the daemon SSL setup similarly to the setup for your WebSphere Application Server. Modify the customization job commands generated in BBOCBRAK (or HLQ.DATA(BBODBRAK) on WebSphere Application Server Network Deployment) to perform the steps in Setting up a Keyring for use by WebSphere Application Server for z/OS.

SSL considerations for WebSphere Application Server administrators:

The Resource Access Control Facility (RACF) customization jobs create an SSL Keyring owned by the WebSphere Application Server for z/OS administrator containing the digital certificate needed to communicate with WebSphere Application Server. However, additional customization is required for administration by other MVS user IDs.

Note that the MVS user ID in the description below is the MVS user ID under which the `wsadmin.sh` process is running, not the user ID specified in the wsadmin request.

In the example below:

- `yyyyy` is the user ID of the new WebSphere Application Server for z/OS administrator
- `xxxxx` is the name of the keyring specified in soap.client.props
- `zzzzz` is the label name used in the BBOSBRAK jobs to specify which certificate authority certificate was used to generate server keys

1. If the new administrator is not a member of the WebSphere Application Server for z/OS administrative group, make sure that the new user ID has access to the appropriate RACF keyrings and digital certificates. For example:

   ```
   PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(yyyyy) ACC(READ)
   PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(yyyyy) ACC(READ)
   ```

2. Use the setup completed by the customization jobs as a model for the additional steps. This information is in the BBOCBRAK member of the <HLQ>.DATA data set generated during the customization process. The BBOCBRAK job contains the set of RACF commands that were used:

   ```
     /* Generating SSL keyrings for WebSphere administrator                  */
   RACDCERT ADDRING(xxxxx) ID( yyyyyy )
     /* Connect WAS CA Certificates to Servers keyring                      */
   "RACDCERT ID(yyyyy) CONNECT (RING(xxxxxx) LABEL('zzzzzzz')   CERTAUTH"
   SETROPTS RACLIST(FACILITY) REFRESH"
   ```

Setting up SSL connections for Java clients:

To configure SSL for use between Java clients running on a workstation and the WebSphere Application Server for z/OS Java 2 Platform, Enterprise Edition (J2EE) server:

1. Determine what SSL repertoire the server is using. For example: `WASKeyring`.
2. Determine the user ID the server is running. For example: `CBSYMSR1`.
3. Export the certificate authority from RACF. For example: `RACDCERT CERTAUTH EXPORT(LABEL('WebSphereCA')) DSN('IBMUSER.WAS.CA') FORMAT(CERTDER)`
4. Move the file to the workstation. (Note that the FTP transfer must use binary.) For example: `c:\tmp` directory
5. Add the digital certificate to the TrustStore used by the client. For example: DummyClientTrustFile.jks file: `keytool -import -file c:\tmp\IBMUSER.WAS.CA -keystore DummyClientTrustFile.jks]`

Setting permission for files created by applications:

Files created by applications running in the servant will have permission bits set according to the default umask. To change the default umask for the servant, specify the _EDC_UMASK_DFLT environment variable in the JCL procedure for the servant. Deployment manager and application servers require group read/write access to the data in their config root.

Deployment manager and application servers require group read/write access to the data in their config root. The server must run with a `007` umask in order to support system management functions. Do not change this umask setting and your server will function correctly.

On the JCL EXEC statement, specify:

```
PARM='ENVAR("_EDC_UMASK_DFLT=xxx")
```

where xxx is the umask value to use (which is 007).

Recommendation: A umask value of 007 will cause files to be created with permission bits set to 770. This is the value recommended by IBM.

Note: See the following documents for more information:
- *z/OS Language Environment Programming Reference*, for more information on ENVAR
- *z/OS C/C++ Programming Guide*, for more information on how to change the UMASK defaults
- *z/OS UNIX System Services Command Reference*

Security auditing: Security auditing is handled in the usual way by the security product. WebSphere Application Server for z/OS uses the System Authorization Facility (SAF), which provides an auditing mechanism consistent with other functions in z/OS or OS/390.

Setting up RACF protection for DB2:

You can use the Resource Access Control Facility (RACF) DSNR resource class to protect DB2 resources. This helps you centralize security management. This section gives you pointers to general information about setting up RACF protection for DB2 and specific information about the resources, groups, user IDs, and permissions used by WebSphere Application Server for z/OS.

There are three functional areas in RACF to consider regarding protection for DB2:
- RACF DSNR class

  The RACF DSNR class controls access to the DB2 subsystems. If the DSNR class is active, then WebSphere Application Server for z/OS controllers and servants need access to the *db2_ssn*. RRSAF profiles, where *db2_ssn* is your DB2 subsystem name. If a controller or servant does not have access, then that region will not initialize.
- Secondary authorization IDs

  DB2 identification and signon exits (DSN3@ATH and DSN3@SGN) are used to assign authorization IDs. If you want to use secondary authorization IDs (RACF group names), then you must replace the default exits with these two sample routines. For details on how to install these sample routines, see *DB2 Administration Guide*.
- Grant statements

  WebSphere Application Server for z/OS does not support the protection of DB2 objects through the DSNX@XAC exit. To protect DB2 objects, you must use GRANT statements.

Steps for defining DB2 options for RACF:

You must complete general tasks for enabling Resource Access Control Facility (RACF) protection for your DB2 system. This includes adding entries to the RACF router table, installing identification and signon exits, and defining RACF user IDs for DB2 started tasks. You must also have your copy of the BBOCBRAJ sample provided with WebSphere Application Server for z/OS.

Perform the following steps to define DB2 resources and authorizations in RACF:
1. Remove the comment marks that surround the REXX and RACF commands. As shipped, the DSNR profile section is commented out.
2. Copy the BBOCBRAJ job to a new file.
3. Submit the job from a user ID with RACF SPECIAL authority.

You know you are done when the job completes successfully.

Understanding System Authorization Facility profile names generated by the Customization Dialog: The WebSphere Application Server for z/OS Customization Dialog generates jobs that help you create the necessary System Authorization Facility (SAF) profiles, such as STARTED, CBIND, or SERVER, that enable your server to run. This article helps you understand how to work with these profiles and determine if you need to also create your own.

At runtime, normal SAF specific and generic profile matching uses a combination of the cell short name, cluster short name (or cluster transition name for a non-clustered server), and server short name to select the appropriate matching profile. See Global security settings for more information on SAF profiles.

WebSphere Application Server for z/OS customization uses two schemes, specific and generic, in the creation of SAF profiles:

- With the specific profile scheme, a set of fully-qualified, specific profiles are created to exactly match the short names that apply to the server you customize. This is either an application server or deployment manager.
- With the generic profile scheme, a set of generic profiles are also created. (For example, the STARTED class BBO*.* profiles.) The purpose of these generic profiles is to provide a default profile for any server that is created administratively and that has a default name so that the servers can operate successfully by default.

  Examples:

  – An application server created through the administrative console has a default server short name of BBOSnnn and a cluster short name (or cluster transition name for a non-clustered server) of BBOCnnn, where nnn is a unique number. By default, this server can start using the BBO* generic profiles.

  – Node federation creates a node agent server. If the base application server you federate is configured with a Java Message Service (JMS) integral provider, then a standalone JMS server is also created. The node agent has a default name of BBONnnn and the JMS server is BBOJnnn, where nnn is a unique number. By default, these servers can start using the BBO* generic profiles.

The generic profiles that customization creates are not required and exist only for your convenience in case you use the default server short names and cluster short names (or cluster transition names for non-clustered servers) generated by WebSphere Application Server for z/OS. You may choose to delete the generic profiles if, for example, your organization has particular naming conventions and you will not use the default names generated by WebSphere Application Server for z/OS. In that case, ensure that you have your own strategy for creating the required SAF profiles, either generic or specific, with your own naming convention--WebSphere Application Server for z/OS does not create them for you.

PropFilePasswordEncoder command reference

Purpose

The **PropFilePasswordEncoder** command encodes passwords located in plain text property files. This command encodes both Secure Authentication Server (SAS) property files and non-SAS property files. After you have encoded the passwords, note that a decoding command does not exist. To encode passwords, you must run this command from the *install_dir*/bin directory of a WebSphere Application Server installation.

Syntax

The command syntax is as follows:

```
PropFilePasswordEncoder file_name
```

Parameters

The following option is available for the PropFilePasswordEncoder command:

-sas
Encodes SAS property files.

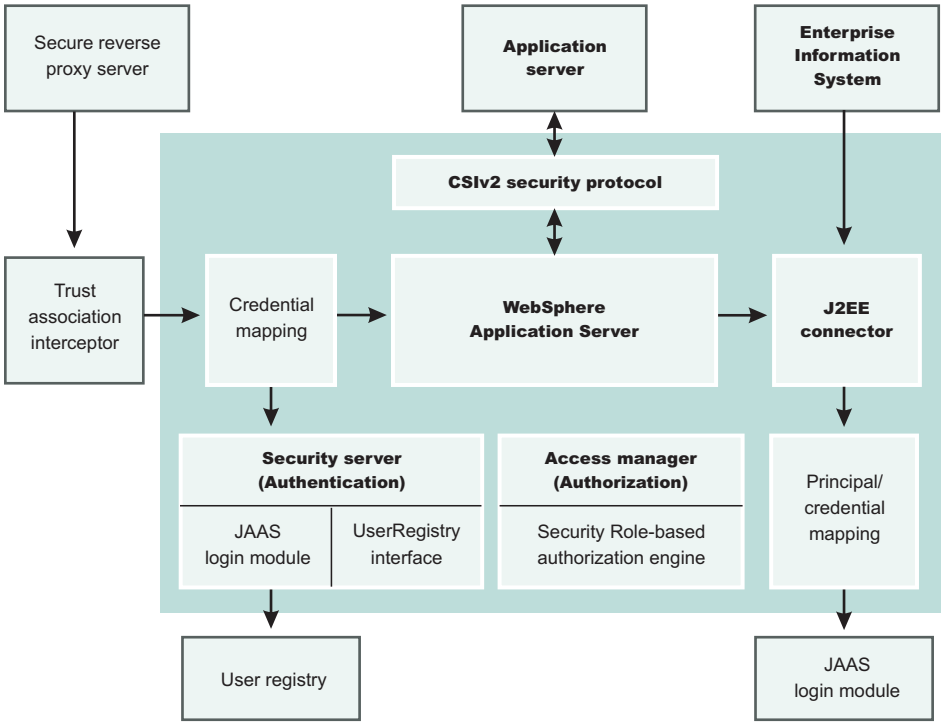
The following examples demonstrate the correct syntax.

```
PropFilePasswordEncoder file_name password_properties_list
PropFilePasswordEncoder file_name -SAS
```

Integrating IBM WebSphere Application Server security with existing security systems

WebSphere Application Server plays an integral part of the multiple-tier enterprise computing framework. WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components to provide end-to-end security. WebSphere Application Server plug-in points are based on standard Java 2 Platform, Enterprise Edition (J2EE) specifications wherever applicable. The WebSphere Application Server development team is actively involved in various standard bodies to externalize and to standardize plug-in interfaces.

In the following example, several typical multiple-tier enterprise network configurations are discussed. In each case, various WebSphere Application Server plug-in points are used to integrate with other business components. The discussion starts with a basic multiple-tier enterprise network configuration:



Terminology

A list of terms used in this discussion follows:

Protocol firewall

Prevents unauthorized access from the Internet to the demilitarized zone. The role of this node is to provide the Internet traffic access only on certain ports and to block other IP ports.

Note: Firewalls can be used to create demilitarized zones, which serve as machines that are isolated from both the public Internet and other machines in the configuration. This improves portal security, especially for sensitive back-end resources such as databases.

WebSphere Application Server plug-in

Redirects all the requests for servlets and JavaServer Pages (JSP) pages. Also referred to in WebSphere Application Server literature as the *Web server redirector*, it was introduced to

separate the Web server from the application server. The advantage of using Web server redirector is that you can move an application server and all the application business logic behind the domain firewall.

Domain firewall
Prevents unauthorized access from the demilitarized zone to an internal network. The role of this firewall is to allow the network traffic originating from the demilitarized zone and not from the Internet.

Directory
Provides information about the users and their rights in the application. The information can contain user IDs, passwords, certificates, access groups, and so forth. This node supplies the information to the security services like authentication and authorization service.

Enterprise information system
Represents existing enterprise applications and business data in back-end databases.

The Web server plug-in

WebSphere Application Server provides the infrastructure to run application logic and communicate with the internal back-end systems and database that web applications and enterprise beans can access. WebSphere Application Server has a built in HTTPS server that can accept client requests. A typical configuration, however, places WebSphere Application Server behind the domain firewall for better protection. A WebSphere Application Server plug-in to the Web server configuration can redirect Web requests to WebSphere Application Server. WebSphere Application Server provides plug-ins for many popular Web servers.

You can configure WebSphere Application Server and the Web server plug-in to communicate through secure SSL channels. You can configure a WebSphere Application Server HTTP server to open communication channels only with a restricted set of Web server plug-ins.

The WebSphere Application Server plug-in routes HTTP requests according to the virtual host and port configuration and URL pattern matching. Client authentication and finer grained access control are handled by WebSphere Application Server behind the firewall.

Tivoli WebSEAL

In cases where the Web server can contain sensitive data and direct access is not desirable, the following configuration uses Tivoli WebSEAL to shield a Web server from unauthorized requests. WebSEAL is a Reverse Proxy Security Server (RPSS) that uses Tivoli Access Manager to perform coarse-grained access control to filter out unauthorized requests before they reach the domain firewall. WebSEAL uses Tivoli Access Manager to perform access control.

User registry implementations

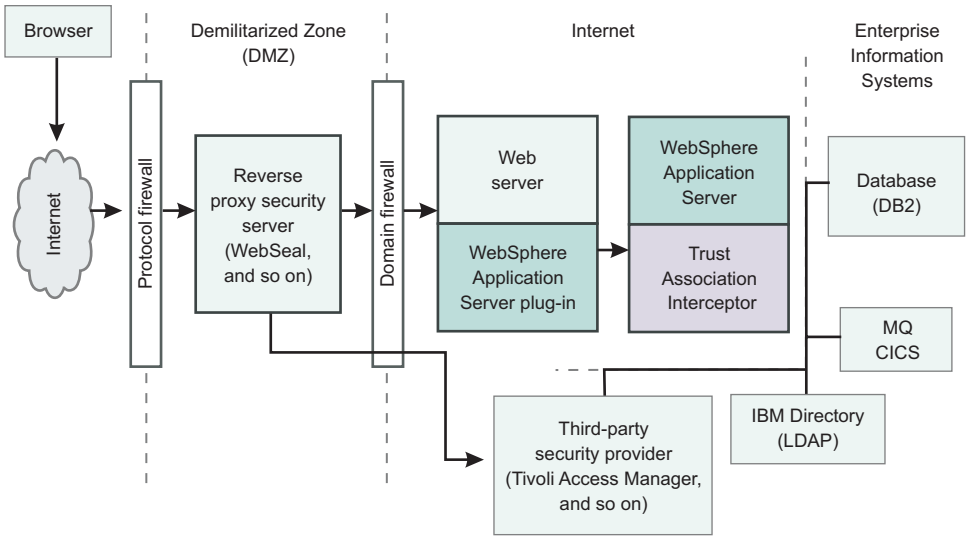
WebSphere Application Server supports various user registry implementations through the pluggable user registry interface.

Note: You can update the WEB_INBOUND System Login Configuration entry with a custom JAAS login module that can map the client user ID from the TAI plug-in to a WebSphere Application Server user ID. You also can change the WEB_INBOUND System Login Configuration entry to map the principal or subject from the TAI plug-in to a user ID identified in the WebSphere Application Serve user registry. In addition to providing identity mapping facilities, security attributes generated by other security systems can be propagated using WebSphere Application Server runtime. For more information, refer to “Enabling security attribute propagation” on page 1056.

WebSphere Application Server also supports users in developing their own custom registry and plug-in through the pluggable user registry interface. When integrated with a third party security provider, WebSphere Application Server can share the user registry with the third-party security provider. In the

particular example of integrating with WebSEAL, you can configure WebSphere Application Server to use the LDAP user registry, which can be shared with WebSEAL and Tivoli Access Manager. Moreover, you can configure WebSphere Application Server to use the Lightweight Third Party (LTPA) authentication mechanism, which supports the Trust Association Interceptor plug-in point.

Basically, the RPSS performs authentication and adds proper authentication data into the request header and then redirects the request to Web server. A trust relationship is formed between an RPSS and WebSphere Application Server, and the RPSS can assert client identity to WebSphere Application Server to achieve single signon (SSO) between RPSS and WebSphere Application Server. When the request is forwarded to WebSphere Application Server, WebSphere Application Server uses the TAI plug-in for the particular RPSS server to evaluate the trust relationship and to extract the authenticated client identity. WebSphere Application Server then maps the client identity to a WebSphere Application Server security credential. For instructions on setting up a trust association interceptor, refer to “Trust associations” on page 924, “Configuring trust association interceptors” on page 928.



When configured to use the LDAP user registry, WebSphere Application Server uses LDAP to perform authentication. The client ID and password are passed from WebSphere Application Server to the LDAP server. You can configure WebSphere Application Server to set up an SSL connection to LDAP so that passwords are in the clear. To set up an SSL connection from WebSphere Application Server to the LDAP server, refer to “Configuring Secure Sockets Layer for the Lightweight Directory Access Protocol client” on page 1186.

J2EE Connector Architecture (J2CA)

WebSphere Application Server supports the J2EE Connector Architecture (J2CA or JCA can be used to abbreviate J2EE Connector Architecture, but this documentation will use J2CA). The connector architecture defines a standard interface for WebSphere Application Server to connect to heterogeneous enterprise information systems (EIS). Examples of EIS includes database systems, transaction processing (such as CICS), and messaging (such as Message Queue (MQ)). The EIS implementation (environments, servers and monitors) can perform authentication and access control to protect business data and resources. Resource Adapters authenticate EIS. The authentication data can be provided either by application code or by WebSphere Application Server. WebSphere Application Server provides a principal mapping plug-in point. A principal mapping module plug-in maps the authenticated client principal to a password credential, (that is, user ID and password, for the EIS security domain). WebSphere Application Server ships a default principal mapping module, which maps any authenticated client principal to a configured pair of user IDs and passwords.

When using some local EIS connectors, WebSphere Application Server for z/OS provides facilities to assign the J2EE user ID as the owner of a connection. For more information, refer to Connection thread identity.

Each connector can be configured to use a different set of IDs and passwords. For a description on how to configure J2CA principal mapping user IDs and passwords, refer to “Managing J2EE Connector Architecture authentication data entries” on page 1035.

Mapping modules

A principal mapping module is a special purpose Java Authentication and Authorization Service (JAAS) login module. You can develop your own principal mapping module to fit your particular business application environment. For detailed steps on developing and configuring a custom principal mapping module, refer to the articles, Developing your own J2C principal mapping module and “Configuring application logins for Java Authentication and Authorization Service” on page 1005.

Resource Access Control Facility (RACF)

Use Resource Access Control Facility (RACF) to take advantage of MQ profiles and run an application that uses Java Message Service (JMS). Using a System Authorization Facility (SAF) mapped user ID, you can issue JMS application programming interfaces (API) to:

- Connect to a queue manager
- Put and get from a queue

Network communication using Secure Sockets Layer and the Transport Channel Service

To fully support the required communications for WebSphere Application Server, a secure communication mechanism is required to ensure that applications are Communicating securely. Configure the Secure Sockets Layer (SSL) channel as part of the transport channel service to provide secure communication for all users.

The SSL channel is a protocol channel providing the same interface as the Transmission Control Protocol (TCP) channel. The SSL channel implements the same application interface that the TCP channel implements so upstream channels can be written to use only TCP channel functions. (SSL function can be provided using the SSL channel without modifying the upstream channel.) The SSL channel communicates with the network using a downstream channel that implements the TCP channel interface.

When the SSL channel is constructed, its initialization parameters provide the information required to use Java Secure Socket Extension (JSSE) services. The SSL channel:

- Uses JSSE APIs to perform security functions
- Uses the JSSE security provider to obtain a configured key store
- Loads the key store from the configured key store name using the configured key store password

The SSL channel receives configuration information from JSSE repertoires configured and maintained by WebSphere Application Server. The SSL channel configuration attribute for the security repertoire name provides a reference to all the security attributes required to initialize the SSL channel. If a security repertoire is not available, channel data can be filled in with a map of the property names. If properties are specified in addition to the repertoire name, they override the parameters in the repertoire. Additional security information can be provided as part of individual container configuration.

Note: 6.1+ Although WebSphere Application Server for z/OS supports System SSL (SSSL) repertoires and JSSE repertoires, SSSL repertoires cannot be used with the SSL channel. Only JSSE repertoires can be used with the SSL Channel.

For more information on the SSL channel, refer to Transport chains or Transport protocol for a high availability manager.

For general information on the SSL, refer to Secure Sockets Layer.

Security considerations for WebSphere Application Server for z/OS

Functions supported on WebSphere Application Server for z/OS

WebSphere Application Server for z/OS supports the following functions.

Table 13. Functions supported on WebSphere Application Server for z/OS

Function	Additional information
RunAs EJB	For more information, see “Delegations” on page 1249.
RunAs for Servlets	For more information, see “Delegations” on page 1249.
SAF-based IIOP Protocols	For more information, see “Common Secure Interoperability Version 2 and Security Authentication Service client configuration” on page 1162.
z/OS connector facilities	For more information, see “Resource Recovery Services (RRS)” on page 504.
Global security enable or disable	For more information, see “Enabling global security” on page 857 and “Disabling global security” on page 847.
SAF keyrings	For more information, see “Using System Authorization Facility keyrings with Java Secure Sockets Extension” on page 900.
Authentication functions	*Authentication function examples*: Basic, SSL digital certificates, form-based login, security constraints, trust association interceptor
J2EE security resources	For more information, see “Securing applications and their environments” on page 825.
Web authentication (LTPA)	For more information, see “Steps for selecting LTPA as the authentication mechanism” on page 918.
IIOP using LTPA	For more information, see “Lightweight Third Party Authentication” on page 919.
WebSphere application bindings	WebSphere application bindings can be used to provide user to role mappings.
Synch to OS Thread	For more information, see “Synchronizing a Java thread identity and an operating system thread identity” on page 890.
J2EE role-based naming security	For more information, see Java 2 Platform, Enterprise Edition (J2EE) specification.
J2EE role-based administrative security	For more information, see Java 2 Platform, Enterprise Edition (J2EE) specification.
SAF registries	For more information, see “User registries” on page 951.
Identity assertion	For more information, see Identity assertion.
Authentication protocols	*Example*: z/SAS, CSIV2 For more information, see “Supported authentication protocols” on page 1161.
CSIv2 conformance level ″0″	For more information, see “Planning to secure your environment” on page 826.
J2EE 1.4 compliance	For more information, see Java 2 Platform, Enterprise Edition (J2EE) specification.

Table 13. Functions supported on WebSphere Application Server for z/OS (continued)

Function	Additional information
JAAS programming model WebSphere extensions	For more information, see Web authentication using the Java Authentication and Authorization Service programming model.

All basic WebSphere Application Server provide the following functions:

- **Using RunAs:** Use RunAs to change the identity of a caller, server, or role. This designation is now part of the servlet specification.
- **Support of SAF-based IIOP authentication protocols:** Network Deployment uses Secure Authentication Services (SAS) for IIOP authentication. z/OS has its own version of SAS called z/OS Secure Authentication Services (z/SAS) (with similar functions but different mechanisms), and it handles functions such as local security, Secure Sockets Layer (SSL)-based authorization, digital certificates with System Authorization Facility (SAF) mapping, and SAF identity assertion.
- **SAF-based authorization and RunAs capability:** This allows you to use SAF (EJBROLE) profiles for permission and delegation security information.
- **Support for z/OS connector facilities:** Instead of using an alias where a user ID and password is stored, the ability to propagate local OS identities is supported.
- **SAF keyring support for HTTP and IIOP:** Use SystemSSL for HTTP, IIOP, and SAF key ring support. You can also use JSSE.
- **Authentication functions:** Web Authentication mechanisms such as basic, SSL digital certificates, form-based login, security constraints, and trust association interceptor offer the same functionality in Version 6.0.x as offered in Version 5.
- **Authorization for J2EE resources:** Authorization for J2EE resources employs roles similar to the ones used in Version 4, and these roles are used as descriptors.
- **Security enablement:** Security can be enabled or disabled globally. When the server comes up there is some level of security on, but security is disabled until the administrator sets it up.
- **Web authentication using LTPA and SWAM:** Single-signon using Lightweight Third Party Authentication (LTPA) or Simple WebSphere Authentication Mechanism (SWAM) is supported.
- **IIOP authentication using LTPA:** IIOP authentication using LTPA is supported.
- **WebSphere Application Bindings for Authorization:** WebSphere Application Bindings for Authorization are now supported.
- **Synch to OS Thread:** Application Synch to OS Thread is supported.
- **J2EE role-based naming security:** J2EE roles are used to protect access to the namespace. The new roles and tasks are cosNamingRead, cosNamingWrite, cosNamingCreate, and cosNamingDelete.
- **Role-based administrative security:** The roles delimiting security are:
  - Monitor (least authorization and is read-only)
  - Operator (can do runtime changes)
  - Configurator (can monitor and configuration privileges)
  - Administrator (most authorization)

Comparing WebSphere Application Server for z/OS with other WebSphere Application Server platforms

A key similarity:

- **Pluggable security model:** The pluggable security model can be authenticated in IIOP (CSIv2), Web Trust Authentication, Java Management Extensions (JMX) Connectors, or the Java Authentication and Authorization Service (JAAS) programming model. You must:
  1. Determine which registry is appropriate and what authentication (token) mechanisms are needed
  2. Determine whether or not the registry is local or remote, and what Web authorizations should be used - Web authorizations include Simple WebSphere authentication mechanism (SWAM) and Lightweight Third-Party Authentication (LTPA)

Key differences include:

- **SAF registries:** Local operating system registries provide premium functionality on z/OS because z/OS spans a sysplex rather than a single server. z/OS provides certificate to user mapping, authorization, and delegation functions.
- **Identity assertion:** Use trusted servers or CBIND to get the authorization required for the server doing the assertion. Distributed platform requires a server to be placed in the trusted server list. z/OS requires a server ID to have a specific CBIND authorization. The Assertion types are SAF user ID, Distinguished Name (DN), and SSL client certificate.
- **zSAS and SAS authentication protocols for IIOP clients:** z/SAS differs from SAS because it supports RACF PassTickets. The SAS layer in WebSphere Distributed uses CORBA portable interceptors to implement their Secure Association Service, and z/OS does not.
- **CORBA features:** z/OS does not support CORBA security interfaces including the CORBA current, LoginHelper, Credentials, and ServerSideAuthenticator models. CORBA functions have been migrated to JAAS.
- **Authentication protocols:** CSIv2 is an Object Management Group (OMG) specification for the z/OS Security Server and is automatically enabled when WebSphere security is enabled. This is a three-layered approach involving a transport layer (SSL/TLS) for message protection, supplemental client authentication layer for user ID and password (GSSUP), and security attribute layer used by middle servers (who must be specially authorized to the target server) for identity assertion.

J2EE 1.3 compliance

Being J2EE-compliant involves:
- **CSIv2 conformance level ″0″:** This is an OMG (related to the z/OS Security Server) specification, which is part of what used to be the CORBA support. CSIv2 is automatically enabled when security is enabled.
- **Use of Java 2 security:** There is ″security-enabled″ and ″Java 2 security-enabled″, and the default for Java2 is ″on″. This provides a fine-grained access control that is code-based as opposed to subject-based authorization. Each class belongs to one particular domain. Permissions protected by Java 2 security include file access, network access, sockets, exiting Java virtual machine (JVM), administration of properties, and threads. The ″security manager″ is what Java 2 uses as a mechanism for managing security and enforcing the required protections. Extensions to Java 2 security include use of dynamic policy (permissions resource type-based rather than code-based), use of specific default permissions defined for resources in template profiles, and use of filter files to disable policy.
- **Use of JAAS programming:** JAAS programming includes a standard set of APIs for authentication. JAAS is the strategic authorization and authentication mechanism. IBM Developer Kit for Java Technology Edition Version 1.4.2 WebSphere Application Server shipped with WebSphere Version 6.0.x (but some extensions are supplied).
- **Use of the servlet RunAs function:** WebSphere Application Server on the distributed platforms (not the z/OS platform) refers to this function as ″Delegation Policy″. You can change identity to run as a system, caller, or role (user). This function is now part of the servlet specification. Authentication involves using a user ID and password and then mapping the alias to the appropriate XML file to find the user ID of the RunAs role.

Compliance with WebSphere Network Deployment at the API/SPI level

Compliance with WebSphere Network Deployment at the API/SPI level makes deploying applications from Network Deployment on z/OS easier. Features enhanced or deprecated by Network Deployment are enhanced or deprecated by z/OS. However, this does not mean there is no migration for z/OS customers. Compliance with WebSphere Network Deployment at the API/SPI level includes:
- **WebSphere Application Server extensions to the JAAS programming model:** The authorization model is an extension of the Java 2 security model for JAAS programming (so it works with the J2EE model). Subject-based authorization is performed on authenticated user IDs. Instead of merely logging in with a user ID and password, there is now a login process that includes creating a login context, passing callback handlers that prompt for user ID and password, and logging in. WebSphere Application Server for z/OS supplies the login module, the callback handler to retrieve the necessary data, the callbacks, the WSSubject choice, getCallerSubject, and getRunAsSubject .

- **Use of the WebSphere Application Server security APIs:** z/OS supports WebSphere Application Server security APIs.
- **Use of secure JMX connectors:** JMX connectors can be used with user ID and password credentials. The two connector types are RMI and SOAP/HTTPS (and are for administration). The SOAP connector uses the JSSE SSL repertoires. The RMI connector is subject to the same advantages and restrictions as IIOP mechanisms (such as CSIv2).

Interoperability issues for security

To have interoperability of Security Authentication Service (SAS) between C++ and WebSphere Application Server, use the Common Secure Interoperability Version 2 (CSIv2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP).

To have interoperability of SAS between WebSphere Application Server and WebSphere Application Server for z/OS use the zSAS authentication protocol over RMI-IIOP.

Interoperating with a C++ common object request broker architecture client

You can achieve interoperability between C++ CORBA clients and WebSphere Application Server using the z/OS Secure Authentication Services (z/SAS) protocols. z/SAS supports many of the same functions as Common Secure Interoperability Version 2 (CSIv2), only z/SAS uses a proprietary architecture. z/SAS supports three types of authentication:

- User ID and password authentication
- User ID and password authentication over SSL
- SSL client certificate authentication

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 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 listed below (security_sslKeyring, client_protocol_user, client_protocol_password) in the file.
- Point to the environment file using the fully qualified path name through the environment variable WAS_CONFIG_FILE. For example, in the test shell script `test.sh`, 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

C++ security setting	Description
client_protocol_password	Specifies the password for the user ID.
client_protocol_user	Specifies the user ID to be authenticated at the target server.

C++ security setting	Description
security_sslKeyring	Specifies the name of the RACF keyring the client will use. The keyring must be defined under the user ID that is issuing the command to run the client.

Interoperating with previous product versions

IBM WebSphere Application Server, Version 6.0.x interoperates with the previous product versions such as Version 5.x. Interoperability is achieved using the zSAS security mechanism for localOS and SAF-based authorization.

1. If SSL is configured on a previous product version, your servers must have a basis to establish trust. Using Resource Access Control Facility (RACF), your system can check to ensure that the intermediate server can be trusted (to confer this level of trust, CBIND authorization is granted by administrators to RACF user IDs that run secure system code). System SSL repertoires use a System Authorization Facility (SAF) keyring to retrieve the personal certificate and trust stores. You must connect the trust basis for the server certificates (on the default setup the certificate authority certificate) of the previous version server into the keyring of the WebSphere Application Server for z/OS Version 6.0.x server.
2. Extract and add server certificates into the server key ring file of the previous version.
   a. Open the server key ring file using the key management utility (iKeyman) and extract the server certificate to a file.
   b. Open the server key ring of the previous product version, using the key management utility and add the certificate extracted from WebSphere Application Server Version 6.0.x.
3. Extract and add server certificates into the server key ring file of the previous version.
   a. Open the server key ring file using the key management utility (iKeyman) and extract the server certificate to a file.
   b. Open the server key ring of the previous product version, using the key management utility and add the certificate extracted from the product.
4. Extract and add trust certificates into the trust key ring file of the previous product version.
   a. Open the trust key ring file using the key management utility and extract the trust certificate to a file.
   b. Open the trust key ring file of the previous product version using the key management utility and add the certificate extracted from the product.
5. If single signon (SSO) is enabled, export keys from the product and import them into the previous product version.
6. Verify that the application uses the correct JNDI name. In WebSphere Application Server Version 6.0.x, the enterprise beans are registered with long JNDI names like, `(top)/nodes/`*`node_name`*`/servers/`*`server_name`*`/HelloHome`. Whereas in previous releases, enterprise beans are registered under a root like, `(top)/HelloHome`. Therefore, EJB applications from previous versions perform a lookup on the Version 6.0.x enterprise beans.

   You can also create EJB name bindings that are compatible with the previous version. To create an EJB name binding at the root Version 6.0.x, start the administrative console and click **Environment** > **Naming** > **Naming Space Bindings** > **New** > **EJB** > **Next**. Complete all the fields and enter a short name (for example, `-HelloHome`) as the JNDI Name. Click **Next** and **Finish**.
7. Stop and restart all the servers.
8. Make sure that the correct naming bootstrap port is used to perform naming lookup. In previous product versions, the naming bootstrap port is **900**. In Version 6.0.x, the bootstrap port is **2809**.

Security: Resources for learning

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

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful in all or 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"
- "Programming model and decisions"
- "Programming specifications"
- "Administration"

Planning, business scenarios and IT architecture
- WebSphere Application Server Library
- WebSphere Application Server Support
- WebSphere Application Server Version 5 Security Redbook
- Accessing the Samples (Samples Gallery)

  The technology sample in the WebSphere Application Server Samples Gallery contains several security-related samples including the form login sample and the Java Authentication and Authorization Service (JAAS) login sample.
- WebSphere Application Server security: Presentation series
- WebSphere Application Server V5 advanced security and system hardening

Programming model and decisions
- **Sun Java Secure Socket Extension (JSSE) documentation**:

  Refer to http://www-106.ibm.com/developerworks/websphere/library/techarticles/0403_yu/0403_yu.html?ca=dnp-314#IDACKF3B for information on setting up WebSphere Application Server using Sun Java Secure Socket Extension (JSSE) at runtime.
- **Java 2 security documentation**:IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4
  - Refer to Java 2 Security check permission algorithm.

Programming specifications
- J2EE Specifications
- EJB Specifications
- Servlet Specifications
- Common Secure Interoperability Version 2 (CSIv2) Specification
- Java 2 Platform, Standard Edition, v 1.4.2 API Specification
- Java Authorization Contract for Containers (JSR 115) Specification

Administration
- z/OS WebSphere Application Server V5 and J2EE 1.3 Security Handbook

  This redbook is designed to help application programmers, security administrators, and application and network architects understand the features provided by WebSphere Application Server Version 5.x on the z/OS platform.
- IBM WebSphere V4.0 Advanced Edition Security
- IBM HTTP Server Support and Documentation
- IBM Directory Server Support and Documentation
- IBM developer kits

  This Web site provides access to the IBM developer kits provided by the IBM Centre for Java Technology Development. Using this Web site, you can find various security and diagnostic information

including information on the Federal Information Processing Standard, Java Version 1.4.1, Java Version 1.4.2, the iKeyman tool, and the Public Key Cryptography Standards (PKCS).
- IBM cryptographic hardware devices
- Supported hardware, software and APIs prerequisite Web site
- IBM Education Assistant
- Understanding LDAP - Design and Implementation
- WebSphere security fundamentals
- WebSphere Application Server V6 Migration Guide

Administering security

Administering secure applications requires access to the WebSphere Application Server administrative console. Log in with a valid user ID and password that have administrative access. To administer security, complete these steps:

1. Configure global security. For more information, see “Configuring global security” on page 881.
2. Assign users to administrator roles. For more information, see “Assigning users to administrator roles” on page 910.
3. Assign users to naming roles. For more information, see “Assigning users to naming roles” on page 913.
4. Configure authentication mechanisms. For more information, see “Configuring authentication mechanisms” on page 918.
5. Configure Lightweight Third Party Authentication. For more information, see “Configuring Lightweight Third Party Authentication” on page 920.
6. Configure trust association interceptors. For more information, see “Configuring trust association interceptors” on page 928.
7. Configure single signon. For more information, see “Configuring single signon” on page 931.
8. Configure user registries. For more information, see “Configuring user registries” on page 955.
   a. Configure local operating system user registries. For more information, see “Configuring local operating system user registries” on page 958.
   b. Configure Lightweight Directory Access Protocol user registries. For more information, see “Configuring Lightweight Directory Access Protocol user registries” on page 961.
   c. Configure custom user registries. For more information, see “Configuring custom user registries” on page 975.
9. (Optional) Configure z/OS Security Authorization Facility (SAF). For information, refer to:
   - *SAF authorization*: “Local operating system user registry settings” on page 960 and “z/OS System Authorization Facility properties” on page 1010
   - *User identities*: “Identity assertion” on page 1158
   - *System Login Configuration*s: “Updating System Login Configurations to perform a System Authorization Facility identity user mapping” on page 1009System Login Configurations (csec_syslogconfsaf), point to creating
   - *EJBROLE profiles*: “System Authorization Facility for role-based authorization” on page 855
10. Configure Java Authentication and Authorization Service login. For more information, see “Configuring application logins for Java Authentication and Authorization Service” on page 1005.
11. Configure an authorization provider. For more information, see “Configuring a JACC provider” on page 1122. To configure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider, see either “Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility” on page 1132 or “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 1134.
12. Configure the Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols. For more information, see “Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols” on page 1161.

13. Configure Secure Sockets Layer. For more information, see “Configuring Secure Sockets Layer” on page 1184.
14. Configure Java 2 Security Manager. For more information, see “Configuring Java 2 security” on page 1214.
15. **Optional:** Configure security attribute propagation. For more information, see “Security attribute propagation” on page 1052.

Configuring global security

It is helpful to understand security from an infrastructure standpoint 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 global security. The following sections help you make these decisions. Read the following articles before continuing with the security configuration.

- “Global security and server security” on page 899
- Introduction: Security

After you understand the security components, you can proceed to configure global security in WebSphere Application Server.

Attention: There are some security customization tasks required to enable security on WebSphere Application Server for z/OS that require updates to the security server (such as Resource Access Control Facility (RACF)) running on your system. You might need to include your security administrator in this process.

1. Start the WebSphere Application Server administrative console by typing `http://yourhost.domain:9060/ibm/console` after the WebSphere Application Server deployment manager has been started. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative user ID and password.
2. Click **Security** on the navigation menu. Configure the authentication mechanism, user registry, and so on. The configuration order is not important. However, select the **Enable global security** option in the Global Security panel after you have completed all of these tasks. When you first click **Apply** or **OK** and the **Enable global security** option is set, a verification occurs to see if the administrative user ID and password can be authenticated to the configured user registry. If the user registry is not configured, the validation fails.
3. Configure a user registry. For more information, see “Configuring user registries” on page 955. Configure a Local OS, Lightweight Directory Access Protocol (LDAP), or custom user registry and then specify the details about that registry. One of these details common to all user registries is the user ID used for the server. This ID is a member of the chosen user registry, but also has special privileges in WebSphere Application Server. The privileges for this ID and the privileges associated with the administrative role ID are the same. The user ID used for the server can access all protected administrative methods. When you use the Local OS user registry on WebSphere Application Server for z/OS, the user ID for the server is not set using the administrative console, but is set through the STARTED class in z/OS.
4. Configure the authentication mechanism. You can choose either Lightweight Third Party Authentication (LTPA), Integrated Cryptographic Services Facility (ICSF), or Simple WebSphere Authentication Mechanism (SWAM). To get details about configuring LTPA, refer to “Configuring Lightweight Third Party Authentication” on page 920. To get details about configuring ICSF, refer to “Steps for selecting ICSF as the authentication mechanism” on page 918. LTPA and ICSF credentials are forwardable to other machines and, for security reasons, these credentials do expire. This expiration time is configurable.

   Refer to “Configuring single signon” on page 931 if you want single signon (SSO) support, which provides the ability for browsers to visit different product servers without having to authenticate multiple times. For form-based login, you must configure SSO when using LTPA or ICSF.

5. Configure the authentication protocol for special security requirements for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) method invocations from Java clients or from server to server. Choose the Common Secure Interoperability Version 2 (CSIv2) or Secure Authentication Service (SAS) protocol or the z/OS Secure Authentication Service (z/SAS) protocol on the z/OS platform.

   The SAS and z/SAS protocols still provide backward compatibility to previous product releases. For details on configuring CSIv2, SAS, or z/SAS protocols, refer to the “Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols” on page 1161 article.

   Attention: In future releases, IBM will no longer ship or support the z/OS Secure Authentication Service (z/SAS) IIOP security protocol. It is suggested that you use the Common Secure Interoperability version 2 (CSIv2) protocols.

6. Verify the SSL repertoires to be used by WebSphere Application Server. The sample customization jobs generated by the WebSphere Application Server for z/OS customization dialogs generate sample jobs to create SSL key rings that are usable if RACF is your security server. These jobs create a unique RACF certificate authority certificate for your installation with a set of server certificates signed by this certificate authority. The Application Server controller's started task ID has a SAF key ring that includes these certificates. (Similarly in a Network Deployment environment, RACF key rings owned by the deployment manager user ID and the node agent user IDs are created.)

   Note: A RACF key ring is uniquely identified by both the key ring name in the repertoire and the MVS user ID of the server controller process. If different WebSphere Application Server controller processes have unique MVS user IDs, you must be sure that a RACF key ring and a private key are generated even if they share the same repertoire.

   There are two kinds of configurable SSL repertoires:
   - The System SSL repertoire is used for HTTPS and IIOP communication, and are used by the native transports. If you want to use the administrative console after security is enabled you must define a System SSL type repertoire for HTTP and select it. You must define a System SSL repertoire and select if IIOP security requires or supports SSL transport, or if a secure RMI connector is selected for administrative requests.
   - The Java Secure Socket Extension (JSSE) repertoire is for Java-based SSL communications.

   Users must configure a System SSL repertoire to use HTTP or IIOP protocols and a JMX connector must be configured to use SSL. If the SOAP HTTP connector (default) is chosen, a JSSE repertoire must be selected for the administrative subsystem. In a Network Deployment environment, click **System Administration > Deployment Manager > Administration Services > JMX Connectors > SOAP Connector > Custom Properties > sslConfig**.

   A set of SSL repertoires are set up by the z/OS installation dialogs. These dialogs are configured to refer to SAF key rings and to files that are populated by the customization process when generating RACF commands.

Repertoire name	Type	Default use
DefaultSSLSettings	JSSE	SOAP JMX connector, SOAP client
DefaultHTTPS	SSSL	Web container HTTP transport
DefaultIIOPSSL	SSSL	z/SAS and CSIV2
RACFJSSESettings	SSSL	None
RACFJSSESettings	JSSE	None

   No additional action is required if these settings are sufficient for your needs. If your want to create or modify these settings, you must ensure that the keystores to which they refer are created.

   If you do create a new alias for your new keystore and truststore files, change every location that references the SSL configuration alias. The following list provides these locations:

- **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound transport**.
- **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 outbound transport**.
- **Security > Global security**. Under Authentication, click **Authentication protocol > zSAS authentication**.
- **Servers > Application servers** > *server_name*. Under Web container settings, click **Web Container**. Under Additional properties, click **HTTP transports** > *host_name*.
- **Servers > Application servers** > *server_name*. Under Security, click **Server security**. Under Additional properties, click **CSIv2 inbound transport**.
- **Servers > Application servers** > *server_name*. Under Security, click **Server security**. Under Additional properties, click **CSIv2 outbound transport**.
- **Servers > Application servers** > *server_name*. Under Security, click **Server security**. Under Additional properties, click **z/SAS authentication**. Select the appropriate SSL configuration from the SSL settings menu.

7. Click **Security > Global security** to configure the rest of the security settings and to enable security.

   This panel performs a final validation of the security configuration. When you click **OK** or **Apply** from this panel, the security validation routine is performed and any problems are reported at the top of the page. When you complete all of the fields, click **OK** or **Apply** to accept the selected settings. Click **Save** (at the top of the panel) to persist these settings out to a file. If you see any informational messages in red text color, then a problem exists with the security validation. Typically, the message indicates the problem. So, review your configuration to verify that the user registry settings are accurate and that the correct user registry is selected. In some cases, the LTPA configuration might not be fully specified. See the “Global security and server security” on page 899 article for detailed information.

   Enable global security

   This option enables or disables global security. See the “Global security and server security” on page 899 article to learn more about global security. When enabled, security for the entire product domain is enabled. You can change some security attributes at a server-specific level.

   Enforce Java 2 Security

   This option enables or disables Java 2 security access control. See “Configuring Java 2 security” on page 1214 for details on Java 2 security in WebSphere Application Server.

   Use Domain Qualified User IDs

   This option determines if user IDs returned by the J2EE APIs such as getUserPrincipal() and getCallerPrincipal() are qualified within the security domain in which they reside.

   Cache Timeout

   The field is the timeout value of the WebSphere Application Server authentication and validation cache. This value is used to determine when to flush a credential from the cache. Any time that the credential is reused, the cache timeout for that credential is reset to this value. Currently, no way is available to flush the cache or purge specific users from the cache.

   Issue Permission Warning

   When you enable this option, a warning is issued during application installation if an application requires a Java 2 security permission that normally is not granted to an application. WebSphere Application Server provides support for policy file management. A number of policy files exist in WebSphere Application Server; some of the policy files 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 related 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 an application should not have according to the J2EE 1.3 specification. For more information on permissions, see the “Java 2 security policy files” on page 1217 (Dynamic Policy) article.

   Active Protocol

   This selection is the active authentication protocol for the object request broker (ORB). RMI/IIOP requests use this protocol to gather security information in a format that both client and server understands. In step 5, you already might have configured one or both of these

authentication protocols. Select **BOTH**, if you need to communicate with versions of WebSphere Application Server prior to Version 5. Select **CSI**, if you only need to communicate with WebSphere Application Server Version 5 or Version 6 servers.

Active Authentication Mechanism
This selection determines which authentication mechanism WebSphere Application Server for z/OS uses. WebSphere Application Server for z/OS Version 6 supports the following authentication mechanisms: Simple WebSphere Authentication Mechanism (SWAM) or Lightweight Third Party Authentication (LTPA), which is the preferred.

Active User Registry
This option indicates the user registry you that chose in step 3. The "Configuring user registries" on page 955 article provides the necessary steps to configure the user registry.

Use the Federal Information Processing Standard (FIPS)
This option enables the FIPS-compliant Java cryptography engine.

8. Save the configuration for the deployment manager to use after the WebSphere Application Server is restarted, if you have selected **OK** or **Apply** on the **Security** > **Global security** panel, and no validation problems occurred.

   "Enabling global security" differs from a stand-alone base application server. In the Network Deployment environment, the configuration is stored temporarily in the deployment manager until it is synchronized with all of the node agents. To save the configuration, click **Save** in the menu bar at the top of the panel.

   Verify that all of the node agents are up and running in the domain. It is recommended that you stop all application servers during this process. If any of the node agents are down, run a manual file synchronization utility from the node agent machine to synchronize the security configuration from the deployment manager. Otherwise, the malfunctioning node agent does not communicate with the deployment manager after security is enabled on the deployment manager.

Enabling global security:

You can decide whether to enable IBM WebSphere Application Server security. You must enable global security for all other security settings to function.

Note: WebSphere Application Server uses cryptography to protect sensitive data and 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 have been configured for the Web Services application.

WebSphere 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 improve performance.

WebSphere Application Server Version 6 provides a SDK that contains strong, but limited jurisdiction policy files. You can download the unrestricted policy files for the Windows, Linux, HP-UX, Solaris, and AIX platforms from the following Web site: IBM developer kit: Security information. Complete the following steps to download and install the new policy files:

1. Click **Java 1.4.2**
2. Click **IBM SDK Policy files**.
   The Unrestricted JCE Policy files for SDK 1.4 Web site is displayed.
3. Click **Sign in** and provide your IBM.com ID and password.
4. Select **Unrestricted JCE Policy files for SDK 1.4.2** 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 global security in WebSphere Application Server. For more information, see “Configuring global security” on page 881. It is important to click **Security > Global security** and select the **Enable global security** option and to save the configuration has been saved to the repository. Verify that the validation that occurs after you click **OK** in the **Security > Global security** panel is successful before continuing. If the validation is not successful and you continue with these steps, you risk the server not starting. Reconfigure the security settings until validation is successful.
2. Push a copy of the new configuration to all of the running node agents using the administrative console. If a node agent fails to get the security-enabled configuration, communication with the deployment manager fails due to a lack of access (the node agent will not be security enabled). To force synchronize a specific node, complete the following steps from the administrative console:
   a. Go to **System administration > Nodes** and select the option next to all the nodes (you do not need to select the deployment manager node).
   b. Click **Full resynchronize** to verify that the file synchronization has occurred. The message might indicate that the nodes already are synchronized. This message is OK. When synchronization is initiated, verify that the Synchronized status displays for all nodes.
3. Stop the deployment manager. Manually restart the deployment manager from the command line or service. To stop the deployment manager, complete the following step:
   a. Go to **System administration > Deployment manager** and click **Stop**. This action logs you out of the administrative console and stops the deployment manager process.
4. Restart the deployment manager process. To restart the deployment manager process, locate the *install_root*`/bin` directory and type the following code:

   ```
   START dmgr_proc_name,JOBNAME=server_short_name,
   ENV=cell_short_name.node_short_name.server_short_name
   ```

   Note: You must enter the previous command on a single line. It is split here for display purposes.

   After the deployment manager initialization is complete, go back into the administrative console to complete this task. Remember that security now is enabled in only the deployment manager. If you enabled single signon (SSO), specify the fully qualified domain name of your Web address, for example, `http://myhost.domain:9060/ibm/console`. When you are prompted for a user ID and password, type the one that you entered as the administrator ID in the configured user registry.
5. If the deployment manager does not start after enabling security, disable security using a script and restart. Disable security by issuing the following command from the `DeploymentManager/bin` directory: `/wsadmin.sh -conntype NONE`. At the prompt enter `securityoff`.
6. Restart all node agents to make them security enabled. You must have restarted the deployment manager in a previous step before completing this step. If the node agent is security-enabled before the deployment manager is security-enabled, then the deployment manager cannot query the node agent for status or give the node agent commands. To stop all node agents, complete the following steps:
   a. Go to **System administration > Node agents** and select the option beside all node agents. Click **Restart**. A message similar to the following example is displayed at the top of the panel: `The node agent on node NODE NAME was restarted successfully`.
   b. Alternatively, if you previously did not stop your application servers, restart all of the servers within any given node by clicking **System administration > Node agents** and by clicking the node agents where you want to restart all the servers. Then, click **Restart all Servers on Node**. This action restarts the node agent and any started application servers.

7. If any node agent fails to restart, perform a manual resynchronization of the configuration. This step consists of going to the physical node and running the client `syncNode` command. This client logs into the deployment manager and copies all of the configuration files to the node agent. This action ensures that the configuration is security-enabled. To resynchronize, complete the following steps:
   a. If the node agent is started, but is not communicating with the deployment manager, stop the node agent by issuing a `stopServer`

Global security settings:

Use this page to configure security. When you enable security, you are enabling security settings on a global level.

To view this administrative console page, click **Security > Global security**.

If you are configuring security for the first time, complete the steps in the ″Configuring server security″ article in the documentation to avoid problems. When security is configured, validate any changes to the registry or authentication mechanism panels. Click **Apply** to validate the user registry settings. An attempt is made to authenticate the server ID to the configured user registry. Validating the user registry settings after enabling global security can avoid problems when you restart the server for the first time.

Enable global security:

Specifies whether to enable global security for this WebSphere Application Server domain.

This flag is commonly referred to as the *global security flag* in WebSphere Application Server information. When enabling security, set the authentication mechanism configuration and specify a valid user ID and password in the selected user registry configuration.

If you have problems such as the server not starting after enabling security within the security domain, then you should resynchronize all of the files from the cell to this node. To resynchronize files, run the following command from the node: `syncNode -username` *`your_userid`* `-password` *`your_password`*. This command connects to the deployment manager and resynchronize all of the files.

If your server does not restart after you enable global security, you can disable security. Go to your `$`*`install_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 security disabled to check any incorrect settings through the administrative console.

Local OS user registry users: When you select **Local OS** as the active local operating system user registry, you do not need to supply a password in the user registry configuration.

Default: Disable

Enforce Java 2 Security:

Specifies whether to enable or disable Java 2 security permission checking. By default, Java 2 security is disabled. However, enabling global security automatically enables Java 2 security. You can choose to disable Java 2 security, even when global security is enabled.

When the **Enforce Java 2 security** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, then 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 have all the required permissions. Consult the WebSphere Application Server documentation and review the Java 2 Security and Dynamic Policy sections if you are unfamiliar with Java 2 security.

Default: Disabled

Enforce fine-grained JCA security:

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 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 **Enforce fine-grained JCA security** 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 Java 2 Security and the **Enforce fine-grained JCA security** option is enabled.

Default: Disabled

Use domain-qualified user IDs:

Specifies that user names returned by methods are qualified with the security domain in which they reside.

This field enables or disables qualifying user names with the security domain ID.

Default: Disabled

Cache timeout:

Specifies the timeout value in seconds for security cache. This value is a relative timeout.

If WebSphere Application Server security is enabled, the security cache timeout can influence performance. The timeout setting specifies how often to refresh the security-related caches. When the cache timeout expires, all cached information becomes invalid.

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.

Data type: Integer
Units: Seconds
Default: 600
Range: Greater than 30 seconds

Issue permission warning:

Specifies that during application deployment and application start, the security run time issues a warning if applications are granted any custom permissions. Custom permissions are permissions defined by the user applications, not Java API permissions. Java API permissions are permissions in `package java.*` and `javax.*`.

WebSphere 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. There is no code base defined or relative code base 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 an application should not have according to the J2EE 1.3 specification. For more information on permissions, see the ″Java 2 security policy files″ article in the documentation.

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.

Prior to version 5.x, the z/OS Security Authentication Service (z/SAS) protocol on z/OS was the only available protocol.

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.

If some servers are version 4.x servers, specify `CSI and zSAS`.

Default: BOTH
Range:
Range: CSI and zSAS, CSI

Active authentication mechanism:

Specifies the active authentication mechanism when security is enabled.

In WebSphere Application Server Network Deployment, Version 6.0.x, the active authentication mechanism is not configurable. Also, this version of the product only supports LTPA authentication.

WebSphere Application Server for z/OS, Version 5.x and later supports the following authentication mechanisms: Simple WebSphere Authentication Mechanism (SWAM), Lightweight Third Party Authentication (LTPA), and Integrated Cryptographic Services Facility (ICSF). Only ICSF and LTPA are configurable on WebSphere Application Server for z/OS, Version 5.x and later. SWAM is not configurable.

Default:
Default: LTPA (WebSphere Application Server Network Deployment)
Range:

Default: SWAM
Range: SWAM, LTPA, ICSF

Active User Registry:

Specifies the active user registry, when security is enabled.

You can configure settings for one of the following user registries:
- Local OS

  Specify this setting if you want your configured Resource Access Control Facility (RACF) (or Security Authorization Facility (SAF)-compliant) security server to be used as the WebSphere Application Server user registry.
- LDAP user registry

  The LDAP user registry settings are used when users and groups reside in an external LDAP directory. When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** or **OK** to validate the changes.
- Custom user registry

Default: Local OS (single, stand-alone server or sysplex and root administrator only)

Range: Local OS (single, stand-alone server or sysplex and root administrator only), LDAP user registry, Custom user registry

Use the Federal Information Processing Standard (FIPS):

Enables the Federal Information Processing Standard (FIPS)-compliant Java cryptography engine.
- Does not affect the Secure Sockets Layer cryptography that is performed by WebSphere 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 WebSphere 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 DES, Triple DES, and AES. Although the LTPA keys are backwards compatible with prior releases of WebSphere Application Server, the LTPA token is not compatible with prior releases.

WebSphere Application Server provides a FIPS-approved Java Secure Socket Extension (JSSE) provider called IBMJSSEFIPS. A FIPS-approved JSSE requires the Transport Layer Security (TLS) protocol because it is not compatible with the Secure Sockets Layer (SSL) protocol.

Default: Disabled

Custom Properties: For an existing configuration, there are a number of profiles that you must modify. To modify the profiles, go into the administrative console and click **Security > Global security**. Under Additional Properties, click **Custom properties**.

```
"security.zOS.domainName" value="TESTSYS"
```

You can modify the following global security custom properties:
- `security.zOS.domainType` specifies if there is a security domain used to qualify security definitions. In WebSphere Application Server for z/OS, the values can be specified as *none*, which indicates that Service Access Facility (SAF) security definitions are of the global sysplex scope or *cellQualified*. This indicates that WebSphere Runtime uses the domain name specified in the property `security.zOS.domainName` to qualify SAF security definitions. If the property is not defined, or a value is not set, *none* is assumed. For example: `"security.zOS.domainType" value="cellQualified"`.
- `security.zOS.domainName` is specified if `"security.zOS.domainType" value="cellQualified"`. The value for `security.zOS.domainName` must be an upper case string from 1 to 8 characters in length, which is used to qualify SAF profiles checked for authorization for the server. If a value is specified here and

cellQualified is selected, the name is also used to identify the application name used in the APPL and Passticket profiles. If a value for `security.zOS.domainName` is not specified, the default value is *CBS390*.

The following profiles are affected by this definition are:

- EJBROLE (if SAF authorization)
- CBIND
- APPL
- PASSTICKET

The customization dialog sets up appropriate SAF profiles during customization if the security domain is defined there. Changing the value of the domainType of domainName requires the customer to make appropriate changes in their SAF profile setup, otherwise runtime errors occur. Refer to "Summary of controls" on page 850 for more information on the specific profile updates required for security domainName related customization and the security domain customization panels.

Synchronizing a Java thread identity and an operating system thread identity: Enterprise JavaBeans (EJBs) support a method-level RunAs role specification that associates a Java 2 Platform, Enterprise Edition (J2EE) role with an EJB method invocation. The EJB method executes using the authority associated with the designated security role. The authority is mapped to the designated role using a user identity. Normally, this identity is recognized by Web-based and J2EE run time and is associated with the current dispatch thread. This identity governs access to only those resources and those facilities subject to J2EE security. The actual OS thread identity is unaffected by the EJB RunAs role selection and is typically the identity of the server.

Setting the OS identity thread synchronizes the J2EE role identity and OS thread (SyncToOSThread). This means that the OS thread identity is associated with the J2EE role identity for the duration of the EJB method invocation (application assemblers and deployers associate the RunAs identity with the operating system thread by setting the thread identity to the RunAs identity for specific bean methods). This association means that the caller or security role identity (rather than the server region identity) is used for z/OS system service requests such as access to files and database management systems. Note that the WebSphere Application Server for z/OS J2EE server can be configured to enable or disable this association (or synchronization). The default setting disables the ability to modify the identity on the operating system thread, regardless of the OS thread identity to RunAs identity setting in the deployment descriptor for the installed application. If the application installer does not enable synchronization, any method that sets the RunAs identity to the operating system thread fails with a no_permission error.

Using the administrative console, you can specify options for thread identity synchronization:

Sync to OS Thread Allowed
Specifies whether an application SynchToOSThread is permitted. When this global security option is selected (meaning *true* is specified) the application-specified SyncToOSThread is honored and subsequently carried out by the EJB and Web containers as indicated by EJB and Web application SyncToOSThread specifications. The default is *false* or disabled.

Connection Manager Sync to OS Thread
Specifies whether the connection manager synchronizes the current J2EE principal to the OS thread when a connection is obtained from a resource reference that specifies `res-auth=container`. The default is *false* or disabled.

You can also select the SyncToOSThread support using a method-level extended deployment descriptor (XDD) for Enterprise JavaBeans (EJBs). Enable this support using a distinguished environment entry defined through the EJB or Web application standard deployment descriptor. During assembly or deployment, bind a value to this variable by specifying:

- *True*, which specifies that the J2EE principal or identity should be synchronized to the OS thread for all requests invoked on the EJB or Web application.

- *False* specifies the J2EE principal application or identity should not be synchronized to the OS thread for all requests invoked on the EJB or Web application. This value is the default.

When processing a request, the Web container understands what roles, if any, are required to access the component represented by the input URL. The container validates requestor authentication and that the authenticated user has been granted permission to the required roles. The Web container makes use of the same SAF-based user registry and EJB role profiles as the EJB container to perform this validation. Therefore, you can use the same user registry and role profiles for administering Web applications as you use for Enterprise Beans and J2EE Services. For setting thread identity, possible active user registries include:

- Local OS
- LDAP
- Custom

Application events that modify the thread identity value include:

Initial value when the first method is set
By default, invocations of servlet service methods and EJB business methods implicitly run as caller (RunAsCaller) unless the Run as field of a policy's implicitly run as caller (RunAsCaller) unless the Run as field of a policy's attribute specifies otherwise. EJB client applications always run as server (RunAsServer). Note that for Web applications if no security constraints are specified the application might run with an unauthenticated user ID.

Method delegation changes to the J2EE identity (RunAs Specified)
The connection manager synchronizes the current J2EE identity with the OS thread when obtaining applications from resources references that have container-managed resource authorization (`res-auth=container`). EJB methods marked with SynchToOSThread cause the J2EE role identity to be synchronized to the OS thread.

WSSubject.doAs()
This setting offers flexibility when associating the Subject with remote calls on a thread without having to do a WSSubject.doAs() to associate the subject with the remote action.

Thread identity is temporarily reset on the server in the following situations:

JavaServer Pages (JSP) Compilation
Web container JSP compilation modifies the identity of the server if SyncToOSThread is enabled for the server (`security_EnableSyncToOSThread=1`).

Access of Stateful Backing Store
EJB container stateful session activation changes the identity of the server if SyncToOSThread is enabled. Always access the EJB stateful session backing store using theidentity of the server.

Web application Reloading
When the Web container reloads the Web application, it changes the server identity if SyncToOSThread is enabled for Web applications.

Connection Manager Requests
When the resource reference specifies `res-auth=application`, the thread identity is temporarily set to the identity of the server.

Note: When running with global security enabled it is recommended that you have Java 2 security enabled. Exercise caution when enabling this support because it can cause general z/OS system resources (such as files and sockets) to fall outside the control of the WebSphere Application Server run time and these system resources management to be accessible to identities established through J2EE applications.

Considerations for setting the Synch to OS Thread Allowed option: With the Synch to OS Thread Allowed support:

1. The application developer or assembler requests behavior by setting the special application environment entry `env-entry` in the deployment descriptor:
   `com.ibm.websphere.security.SyncToOSThread = true | false`.
2. The system administrator grants the request made by the application developer or assembler using an application server configuration setting.

You can select the **Synch to OS Thread Allowed** option at development time or at assembly time:

- At development time, use Rational Application Developer to add an environment entry (environment variable) to the Enterprise JavaBean (EJB) component or Web application module. **Important**: Environment entries (environment variables) can be defined on individual EJB components but cannot be set on individual Web components. A Java 2 Platform, Enterprise Edition (J2EE) standard deployment descriptor can be defined for each EJB component and for each Web application module. Note that a Web component is either a servlet or JavaServer Pages (JSP) files. For Web components, environment entries (environment variables) can only be set on a Web application module. A Web application module contains servlets and JSP files.
- At assembly time, you can add or change environment entries (environment variables) using an assembly tool. For more information, see Starting an assembly tool.

WebSphere Application Server for z/OS global security options:

Use this page to determine which global security options to specify for WebSphere Application Server for z/OS.

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

1. Click **Security** > **Global security**.
2. Under Additional properties, click **z/OS security options**.

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. Under Additional properties, click **z/OS security options**.

If you are configuring security for the first time, complete the steps in the Configuring global security article prior to making changes. After 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 to the configured user registry. Validating the user registry settings after enabling global security can reduce potential problems when you restart the server for the first time.

Remote identity:

Specifies the System Authorization Facility (SAF) user ID that is assumed for the Internet Inter-ORB Protocol (IIOP) unauthenticated clients that make requests of this server from another system.

Specifies whether an application remote identity is permitted.

Local identity:

Specifies the SAF user ID that is assumed for the Internet Inter-ORB Protocol (IIOP) unauthenticated clients that make requests of this server from the same system.

Specifies whether an application local identity is permitted.

Support the synchronization of the OS thread:

Indicates if an operating system thread identity is enabled for synchronization with the Java 2 Platform, Enterprise Edition (J2EE) identity that is used in the WebSphere Application Server run time if an application is coded to request this function.

Synchronizing the operating system identity to the J2EE identity causes the operating system identity to synchronize with the authenticated caller, or delegated RunAs identity in a servlet or Enterprise JavaBeans (EJB) file. This synchronization or association means that the caller or security role identity, rather than the server region identity, is used for z/OS system service requests such as access to files.

For this function to be active, the following conditions must all be true:

- The Sync to OS thread allowed value is `true`.
- A WebSphere application includes within its deployment descriptor an env-entry of com.ibm.websphere.security.SyncToOSThread set to `true`.
- The configured registry is local OS.

When these conditions are true, the OS thread identity is initially set to the authenticated caller identity of a Web or EJB request. The OS thread is modified each time the J2EE identity is modified. The J2EE identity can be modified either by a RunAs specification on the deployment descriptor or a programmatic `WSSubject.doAs()` request.

Note: When a servlet is deployed with no security constraints, the OS thread is set to the value of the configured unauthenticated identity property in the Local OS registry definition (`com.ibm.security.SAF.unauthenticated`).

If the Sync to OS thread allowed value is `false`, which is the default setting, the ability to modify the identity on the operating system thread of the deployment descriptor setting in the deployment descriptor of the installed application is disabled. If the server is not configured to accept enable synchronization, and the application deployment descriptor, com.ibm.websphere.security.SyncToOSThread, is set to `true`, a BBOJ0080W warning stating that the EJB requests the SyncToOSThread option, but the server is not enabled for the SyncToOSThread option is issued.

Any J2EE Connector architecture (J2CA) connector that uses the thread identity support must support thread identity. Customer Information Control System (CICS), Information Management System (IMS), and DB2 support thread identity. CICS and IMS support thread identity only if the target CICS or IMS is configured on the same system as the WebSphere Application Server for z/OS. DB2 always supports thread identity. If a connector does not support thread identity, the user identity that is associated with the connection is based on the default user identity that is supported by the particular connector.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Enable the connection manager RunAs thread identity.:

Specifies that the connection manager SyncToOSThread method is supported for applications that specify this option.

When you enable this setting, the method can process a request that modifies the operating system identity to reflect the Java 2 Platform, Enterprise Edition (J2EE) identity. This function is required to take advantage of thread identity support. J2EE Connector architecture (J2CA) connectors that access local resources on a z/OS system can use the thread identity support. A set of J2CA connectors that accesses local z/OS resources defaults to the J2EE identity of the application if all of the following conditions are true:

- Resource authorization is set to container-managed (`res-auth=container`).
- An alias entry is not coded when deploying the application.

- The connection manager Sync to OS thread setting is set to `enabled`.

Any J2CA connector that uses the thread identity support must support thread identity. Customer Information Control System (CICS), Information Management System (IMS), and DATABASE 2 (DB2) support thread identity. CICS and IMS support thread identity only if the target CICS or IMS is configured on the same system as WebSphere Application Server for z/OS. DB2 always supports thread identity. If a connector does not support thread identity, the user identity that is associated with the connection is based on the default user identity that is supported by the particular connector.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Understanding application Synch to OS Thread Allowed: Use application Sync to OS Thread Allowed to synchronize a Java thread identity (or JAAS subject) with the OS thread identity for the duration of the current Java 2 Platform, Enterprise Edition (J2EE) application request. If you do not choose this option the OS thread identity value is the same as the servant identity value. Refer to "Synchronizing a Java thread identity and an operating system thread identity" on page 890 for more information.

Application Sync to OS Thread Allowed requires configuration in both the application and the application server:

1. The WebSphere Application Server developer must configure the application to declare that it wants to execute with application Sync to OS Thread
2. The WebSphere Application Server administrator must configure the application server to enable application Sync to OS Thread Allowed

The J2EE application developer configures the application for individual Enterprise JavaBeans (EJB) or Web applications by setting a special `env-entry` in the deployment descriptor `com.ibm.websphere.security.SyncToOSThread = true | false`. The default case in which this deployment descriptor is not specified is equivalent to defining it with a value of `false`.

When an EJB or Web application that requests Sync to OS Thread Allowed is dispatched, the application server (at the request of the EJB Container or the Web Container) synchronizes the OS thread identity associated with the current Java thread identity so the Java thread identity is current on the native thread. This synchronization is effective as long as the EJB or Web application is running the current request. When the EJB or Web completes processing, the native thread is restored to its former state.

If the application requests Sync to OS Thread Allowed but Sync to OS Thread Allowed is not enabled in the application server, when the application attempts to run a *no permission* exception is issued. If the application does not request Sync to OS Thread Allowed but Sync to OS Thread Allowed is enabled in the application server, no synchronization occurs and the current OS thread identity remains the same as the server identity.

Refer to "Understanding Java 2 Platform, Enterprise Edition identity and an operating system thread identity" on page 896 for more information about the identities discussed above.

Understanding Connection Manager RunAs Identity Enabled and operating system security: **Operating system thread security**: Under certain configurations of J2EE Connector Architecture (JCA), Java Message Service (JMS), or Java database connectivity (JDBC) connectors on WebSphere Application Server for z/OS, the OS thread identity is the identity used to create the enterprise information systems (EIS) connection. Refer to Connection thread identity for more information on which configurations support OS thread security.

WebSphere Application Server includes connector configurations that use operating system thread security. By enabling Connection Manager Sync to OS Thread support, the J2EE identity (the RunAs

identity, for example) can be used to obtain the EIS connection for connector configurations that use operating system thread security. The Connection Manager Sync to OS Thread support is enabled by selecting the **Enable the connection manager RunAs thread identity** option, which is available by clicking **Security > Global security > z/OS security options**. If the **Support the synchronization of the OS thread** option is not enabled on the same administrative console panel, the connection to a resource manager under a connector configuration that uses operating system thread security is obtained using the server identity (which serves as a default in this case). Refer to "WebSphere Application Server for z/OS global security options" on page 892 for more information.

The WebSphere Connection Manager performs the operating system thread security-related functions. The Connection Manager synchronizes the OS thread identity with the Java thread identity (this Java thread identity corresponds to the J2EE identity) before obtaining the EIS connection. Refer to "Synchronizing a Java thread identity and an operating system thread identity" on page 890 for more information. After the Connection Manager performs the synchronization, the OS thread identity is temporarily replaced with the Java thread identity, and the Java thread identity is the identity used to obtain the EIS connection. This means that Connection Manager Sync to OS Thread support provides a way to obtain an EIS connection using the Java thread identity (the RunAs identity, for example). After obtaining the connection the Connection Manager restores the previous OS thread identity.

Note:

- The application Sync to OS Thread Allowed setting is not pertinent to determining which identity is used to create a connection under a connector configuration that supports operating system thread security. Using thread identity support explains which identity is used to create a connection in which the configuration is unchanged by the application Sync to OS Thread Allowed support. In particular, for connector configurations that use operating system thread security (but in which Connection Manager Sync to OS Thread is disabled), the server identity is used to create the connection regardless of the application Sync to OS Thread Allowed setting or the current RunAs identity.
- Connection Manager Sync to OS Thread support is only pertinent to obtaining EIS Connections managed by WebSphere Connection Management. For example Connection Manager Sync to OS Thread support might be pertinent to Java database connectivity (JDBC) Connections obtained from application requests on DataSource objects configured via WAS Admin and then looked up in Java Naming and Directory Interface (JNDI). (This would depend on whether or not a specific DataSource instance under a specific JDBC provider used OS thread security or not). However, Connection Manager Sync to OS Thread support would not be pertinent for JDBC Connections obtained using the unmanaged `DriverManager.getConnection(...)` API. Access to such unmanaged resources for which the authorization is performed against the OS thread identity might be affected by the application Sync to OS Thread Allowed support, however.
- Connection Manager Sync to OS Thread support is used (or not used) for connection requests made by user-written code (such as JMS or JDBC calls from a stateless session bean), connection requests made by certain components of the WebSphere Application Server (such as the Message Driven Beans (MDB) Listener), or connection requests made by tooling-generated code (such as container-managed persistence (CMP) beans).
- Some (but not all) connector configurations that use the J2EE identity also use OS Thread Security. Connector configurations such as the Customer Information Control System (CICS) CTG Connector in local mode allow use of the J2EE identity using a different Connection Manager mechanism to create the EIS connection. This configuration does not use operating system thread security.

Refer to Connection thread identity for information for details of connector configurations that use operating system thread security. You can also refer to Using thread identity support.

Refer to "Understanding Java 2 Platform, Enterprise Edition identity and an operating system thread identity" on page 896 for more information about the identities discussed above.

When to use application Synch to OS Thread Allowed: Specify application Synch to OS Thread Allowed to use the Java thread identity to access the non-WebSphere-managed resources accessed by your application. As a result of exploiting the application Synch to OS Thread Allowed support, access control privileges associated with the current Java thread identity (not the access control privileges for the server identity) are applied when accessing these resources. (An example of a non-WebSphere-managed resource is the file system.)

Use application Synch to OS Thread Allowed to control file system access based on the Java thread identity. The default Java thread identity is the client identity, which is the user who invoked the application. The Java 2 Platform, Enterprise Edition (J2EE) RunAS role deployment descriptor settings can override this default to choose from other choices. These choices include the server identity or the specified role, such as a user ID (chosen by the application server) configured to be in the specified role. By running with the Java thread identity and specifying Synch to OS Thread Allowed, all file system access control decisions are based on the access privileges of the Java thread identity. Refer to “Deploying secured applications” on page 1246 and Developing secured applications for details on WebSphere role-based security.

Application Synch to OS Thread Allowed is not relevant to container managed persistence (CMP) entity beans but Connection Management RunAs Identity Enabled might be relevant, depending on the JDBC Provider. Refer to “Understanding Connection Manager RunAs Identity Enabled and operating system security” on page 894 for more information for CMP entity beans.

Refer to “Understanding Java 2 Platform, Enterprise Edition identity and an operating system thread identity” for more information about the identities discussed above.

When to use Connection Manager RunAs Identity Enabled: Specifying Connection Manager RunAs Identity Enabled allows you to use the security policy of the resource manager to govern access control decisions made when Java 2 Platform, Enterprise Edition (J2EE) clients invoke a WebSphere application accessing the resource managed by that resource manager.

For example, if you have a preexisting DB2 for z/OS security policy that controls which users have access to which tables, you want to have that policy enforced when users access WebSphere applications that also access DB2 for z/OS. The J2EE identity (the client identity by default) rather than the operating system identity (server identity) is used to establish connections to DB2 for z/OS when Connection Manager RunAs Identity Enabled is selected. DB2 for z/OS table access for the application is determined using your preexisting DB2 for z/OS security policy based the application invocation.

Refer to “Understanding Java 2 Platform, Enterprise Edition identity and an operating system thread identity” for more information about the identities discussed above.

Understanding Java 2 Platform, Enterprise Edition identity and an operating system thread identity:
Understanding the different types of identities: A WebSphere Application Server user is identified using an identity that must be authenticated by WebSphere Application Server in order to access a WebSphere Application Server application in a secure environment. The WebSphere Application Server authenticates the user identity and represents the user with a Java Authentication and Authorization Service (JAAS) subject. A subject contains one or more principals (which are technology-dependent representations of the authenticated user identity). More detail follows:

User identities

J2EE identity
The user identity authenticated by WebSphere and used for access control decisions made by the WebSphere Application Server at Java 2 Platform, Enterprise Edition (J2EE) runtime (such as the user identity associated with a J2EE application request and used in EJB method permission access control decisions).

Operating system (OS) identity
The user identity authenticated by the underlying operating system and used for access

control decisions made by the OS and its subsystems (such as the user identity associated with a WebSphere Application Server for z/OS servant by the SAF STARTED class facility and used by the file system for access control decisions when the server attempts to access files).

Thread identity

Java thread identity
The J2EE identity currently associated with a Java thread managed by the WebSphere J2EE runtime (a Java thread is the Java Virtual Machine (JVM) representation of a thread). The Java thread identity is associated with an operating system (OS) thread, but the JVM manages the user identity on the Java representation of the thread - separate from the user identity that the operating system manages on the operating system thread. The J2EE identity is current on the Java thread for the life of the a given application request

OS thread identity
The operating system identity currently associated with the operating system thread. The OS thread identity is typically the user identity assigned to servant and is normally not the same as the Java thread identity. Note that J2EE maintains a J2EE identity that corresponds to the OS thread identity assigned to the servant. This J2EE identity can be used as a RunAs identity.

RunAs identity
The J2EE identity chosen as the Java thread identity for a given J2EE application request (based on the RunAs deployment descriptor policy on an Enterprise JavaBeans (EJB) invoked within the J2EE application request). The J2EE identity is normally the identity of the authenticated user who has made the J2EE application request. WebSphere Application Server RunAs policy allows three choices in assigning the Java thread identity for the current request:

1. Assign the client (for example, user) J2EE identity - also referred to as selecting RunAs of ″Caller″
2. Assign the server's J2EE identity
3. Assign the J2EE identity that is in the specified role

When security is enabled, each WebSphere Application Server for z/OS request that invokes a J2EE component is authenticated to ensure that an authorized user is requesting access. A user is represented by a J2EE identity (also called a JAAS subject). This J2EE identity contains one or more principals, and each principal corresponds to a specific user identity. This association is managed by the WebSphere Application Server. The J2EE identity and operating system OS thread identity are associated with each other because they have the same name and represent the same user.

WebSphere Application Server for z/OS dispatches component requests in one of its available servant processes. Within the servant process the component request is dispatched on a Java thread. A Java thread is then mapped internally by the JVM to a z/OS thread control block (TCB). A TCB is an operating system thread and is considered part of the native process infrastructure. A servant process has a OS identity assigned to it when it starts. The z/OS security policy uses the SAF STARTED class facility to assign the identity.

J2EE authorization decisions including role authorization and permission checking are determined using the J2EE identity. Through a configuration setting, role authorization checking can be delegated to the underlying operating system security manager (such as System Authorization Facility (SAF)), in which case the associated operating system OS identity is used in the role authorization decision.

Some resource managers on z/OS use the OS thread identity to make authorization decisions. For example, file system access control is determined entirely based on which OS thread identity is currently on the TCB when the file is accessed. Similarly, local Java database connectivity (JDBC) connections to DB2 for z/OS use the TCB OS thread identity as the authorization identity under certain configurations. For

resource managers that use the OS thread identity such as DB2 for z/OS (and unlike the file system) that applications access through Java Message Service (JMS), JDBC, or J2EE Connector Architecture (JCA) connectors managed by the WebSphere Application Server for z/OS connection management, we say that the connectors to these z/OS resource managers ″use operating system thread security″. For more information, refer to:

- “Synchronizing a Java thread identity and an operating system thread identity” on page 890
- “Understanding Connection Manager RunAs Identity Enabled and operating system security” on page 894
- “Understanding application Synch to OS Thread Allowed” on page 894
- Connection thread identity
- Using thread identity support

Configuring global security:

The enablement process is divided into two steps. Configuring and enabling global security in the Network Deployment environment differs from a standalone base application server. In the Network Deployment environment, the configuration is stored temporarily on the Deployment Manager until it gets synced up with all of the Node Agents. Also, the Network Deployment environment uses LTPA as the authentication mechanism so that credentials can be forwarded among processes securely. LTPA requires the following additional configuration steps:

1. Configure security so that the right information is provided for global security, which will be propagated to all of the nodes.
2. Enable security on all nodes. This includes ensuring that the files are synchronized and that the processes all get restarted in the correct order. After security is enabled in a process, it cannot accept some commands that have required access rights assigned. Therefore, the order of the processes that get restarted is important.

Complete the following steps to configure global security in the WebSphere Application Server Version 6 environment.

1. Configure the User Registry.
   a. For LocalOS, enter the server’s user ID and password that will be used to authenticate other users and is given administrative privileges for other WebSphere tasks. Make sure the user ID provided has ″Act as Part of Operating System″ privileges in Windows and root privilege in UNIX environments. Click **Apply** or **OK** to save the changes.
   b. For Lightweight Directory Access Protocol (LDAP), enter the server’s user ID and password. Ensure that this user ID is not the LDAP administrative user ID. Enter the LDAP type, host, port, and base distinguished name (DN). These are the required fields. Configure any other LDAP properties as necessary including the Advanced LDAP properties. Remember to click **Apply** or **OK** at each panel to save the changes.
   c. For Custom, enter the server’s user ID and password. Also, enter the class name of the implementation of the custom user registry. This should implement the com.ibm.websphere.security.UserRegistry interface. Click **Apply** or **OK** to save the changes.
2. Configure the LTPA authentication mechanism.
   a. Enter a password for generating LTPA keys. Re-enter the password for validation. Click **Apply** to save the password. Next, press the Generate Keys button to generate a set of keys for use in encrypting LTPA tokens.
   b. Configure Single Signon (SSO). Click on the link below to go to the Single Signon panel. Make sure it is enabled and enter the domain portion of the servers hostname. This is the `austin.ibm.com` portion for a server host of `machine1.austin.ibm.com`. Click **Apply** or **OK** to save the changes.
3. Configure the Global Security panel.

a. Choose which Active User Registry you want to use based on the one you configured above. Change any other attributes on this panel as desired. Click on the enable check box to turn ON global security.
b. Select **Apply** to validate the changes you've made above. If there are any problems reported above in the Messages section, try going back through the configuration to see if there is something that was missed. Verify that the server ID used for the user registry is valid.
c. Do not shut down the Deployment Manager or Node Agents yet. Go to ″Steps to enable global security in ND″ for the correct procedure for allowing this configuration to propagate to all of the nodes in the right sequence.

4. Select **Save** to write the changes out to the repository.

Global security and server security

The term *global security* refers to the security configuration that is effective for the entire security domain. A security domain consists of all the servers configured with the same user registry realm name. On the z/OS platform, the term *global security* refers to the security configuration that is effective for the WebSphere Application Server cell.

For WebSphere Application Server for z/OS, a Local OS registry refers to the Resource Access Control Facility (RACF) (or System Authorization Facility (SAF) compliant) user database configured for the sysplex. Selecting the Local OS registry as the active registry in WebSphere Application Server for z/OS enables you to take advantage of z/OS System Authorization Facility functions directly using the WebSphere principals:

- Share identities with many other z/OS connector services
- Ability to use SAF authorization
- Use of SAF delegation, which minimizes the need to store user IDs and passwords in many locations in the configuration
- Additional audit capabilities

Note that these functions are available using other registries, but require identity mapping to be done through modifications to the WebSphere Application Server system login configuration and JAAS login modules. Refer to “Updating System Login Configurations to perform a System Authorization Facility identity user mapping” on page 1009 for more information.

Configuration of global security for a security domain consists of configuring the common user registry, the authentication mechanism, and other security information that defines the behavior of a security domain. The other security information that is configured includes the following components:

- Java 2 Security Manager
- Java Authentication and Authorization Service (JAAS)
- Java 2 Connector authentication data entries
- Common Secure Interoperability Version 2 (CSIv2) / z/OS Secure Authentication Service (z/SAS) authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security)
- Other miscellaneous attributes.

In a Network Deployment environment, where multiple nodes and multiple servers within a node are possible, you can configure certain attributes at a server level. The attributes that are configurable at a server level include security enablement for the server, Java 2 Security Manager enablement, and CSIv2/SAS (or CSIv2 / z/SAS on the z/OS platform) authentication protocol (RMI/IIOP security). You can disable security on individual application servers while global security is enabled, however, you cannot enable security on an individual application server while global security is disabled.

While application server security is disabled for user requests, administrative and naming security is still enabled for that application server so that the administrative and naming infrastructure remains secure. If cell security is enabled, but security for individual servers is disabled, J2EE applications are not

authenticated or authorized. However, naming and administrative security is still enforced. Consequently, because Naming Services can be called from user applications you need to grant Everyone access to the naming functions that are required so that these functions accept unauthenticated requests. User code does not directly access administrative security except through the supported scripting tools.

Using System Authorization Facility keyrings with Java Secure Sockets Extension:

WebSphere Application Server for z/OS running at maintenance levels before W502000 stored digital certificate information in two different places because of the following Software Development Kit (SDK) restrictions:

- Java Secure Socket Extensions (JSSE) used digital certificates stored in hierarchical file system files
- Secure Sockets Layer (SSL) used digital certificate information stored in the System Authorization Facility (SAF) database

Systems customized at W502000 or above use the single (SAF) digital certificate repository by default, and do not need the modifications described below.

WebSphere Application Server for z/OS customers running server W50100x or later, with Java Development Kit 1.3 level SR20 or later, can modify their WebSphere Application Server systems to use SAF for JSSE as well as SSL (eliminating the need to maintain duplicate certificates in the HFS). The instructions below describe how to enable this support.

Note: Systems customized at maintenance levels at or after W502000 use the single (SAF) digital certificate repository by default, and do not need the modifications described below.

To use SAF certificates with JSSE:

1. Update the Java Management Extensions (JMX) connector settings to indicate the SAF keyring names for the node.
   a. Log in to the administrative console using an identity with administrator authority.
   b. Click **Servers > Application servers >** *server_name*.
   c. Under Server infrastructure, click **Administration > Administration services**.
   d. Under Additional properties, click **JMX connectors**.
   e. On the JMX Connectors panel, click **SOAPConnector**.
   f. Under Additional Properties, click **Custom Properties**.
   g. On the Custom properties page, click **sslConfig**.
   h. On the sslConfig page, look at the Value field. Verify that this field says *node_name/DefaultSSLSettings*, where *nodename* represents the node name where the application server resides. Record the node name for a subsequent step.
   i. Select ***node_name*/RACFJSSESettings** from the list next to the Value field, where *node_name* is the same as the node name that you previously recorded.
   j. Click **OK**. The Custom Properties page appears with a message indicating that changes are made to your local configuration. Do not click **Save** because additional changes that are required.
2. Click **Servers > Application servers** and repeat the previous substeps for each of the other application servers in the cell.
3. Update the Java Management Extensions (JMX) connector settings to indicate the SAF keyring names for the deployment manager node.
   a. Click **System administration > Deployment manager**.
   b. Under Additional properties, click **Administration services > JMX Connectors**.
   c. On the JMX Connectors panel, click **SOAPConnector**.
   d. Under Additional properties, click **Custom properties**.
   e. On the Custom properties page, click **sslConfig**.

f. On the sslConfig page, look at the Value field. This field displays *dmnode/DefaultSSLSettings*, where *dmnode* represents the deployment manager node name. Record the node name for a subsequent step.

g. Select **dmnode/RACFJSSESettings** from the list next to the Value field, where **dmnode** represents the Deployment Manager node name.

h. Click **OK**. After a short time the Custom Properties page appears with a message at the top indicating that changes have been made to your local configuration. Do not click **Save** at this point because there are additional changes that are required.

4. Update the Java Management Extensions (JMX) connector settings to indicate the SAF keyring names for the node agent.

   a. Click **System administration > Node agents >***Node_name*. Record the node agent name for the next step.

   b. Under Additional properties, click **Administration services > JMX Connectors**.

   c. On the JMX Connectors panel, click **SOAPConnector**.

   d. Under Additional properties, click **Custom properties**.

   e. On the Custom properties page, click **sslConfig**.

   f. On the sslConfig page, look at the Value field. This field displays *nodename/DefaultSSLSettings*, where *nodename* is the node name where the node agent resides. Record the node name for a subsequent step.

   g. Select **nodename/RACFJSSESettings** from the list next to the Value field, where **nodename** is the node name that you previously recorded.

   h. Click **OK**. The Custom Properties page is displayed with a message indicating that changes have been made to the local configuration. Do not click **Save** at this point because additional changes are required.

5. Click **System administration > Node agents** and repeat the previous substeps for each of the other node agents servers in the cell.

6. Click **Save** when the *"Changes have been made to your local configuration. Click Save to apply changes to the master configuration"* message is displayed.

7. On the Save page, select the **Synchronize changes with Nodes** option and click **Save**. After the changes are saved, the administrative console returns to the home page.

8. Update the `soap.client.props` file to indicate the SAF keyring names that are appropriate for your configuration. The `soap.client.props` file is used by the `wsadmin.sh` script and is located in the application server or deployment manager `(user.install.root)/properties` file. The purpose of the `soap.client.props` file is to specify the values used by Simple Object Access Protocol (SOAP) clients such as `wsadmin.sh`. In a cell configured before WebSphere Application Server for z/OS maintenance level W502000, the `soap.client.props` file indicates the names of the Java key stores used by JSSE. Once your cell is using SAF keyrings for JSSE administration, verify that SAF keyrings are being used for SOAP clients.

   The `soap.client.props` file is used by the `wsadmin.sh` script.

   Changes to wsadmin client SAF keyrings require updates to the `soap.client.props` file and the creation of a keyring for administrators. Specify the following values:

   ```
   com.ibm.ssl.protocol=SSL
   com.ibm.ssl.keyStoreType=JCERACFKS
   com.ibm.ssl.keyStore=safkeyring:///yourkeyringName
   com.ibm.ssl.keyStorePassword=password
   com.ibm.ssl.trustStoreType=JCERACFKS
   com.ibm.ssl.trustStore=safkeyring:///yourKeyringName
   com.ibm.ssl.trustStorePassword=password
   ```

   The password value specified does not represent a real password because you can use any string. Replace the string `yourKeyringName` with your administrative SAF keyring. The keyring name used by

all WebSphere administrators and the administrative started task user ID (default WSADMSH) must be the same. Additionally, a keyring must be created for each user that uses the wsadmin.sh file with the SOAP connector when using SAF keyrings and security is enabled. (A keyring is created by the customization process for your initial administrative user ID, such as WSADMIN.)

A description of how to create keyrings for administrative users in SAF is described in SSL considerations for WebSphere Application Server administrators.

9. Recycle the cell.

Configuring server security

Note: User Registry properties include SAF properties such as com.ibm.security.SAF.authorization and com.ibm.security.SAF.unauthenticated identities.

You can customize security to some extent at the application server level. You can disable user security on an application server (administrative security remains enabled when global security is enabled). You can also modify Java 2 Security Manager, CSIv2 or z/OS Secure Authentication Services (z/SAS), and some of the other security attributes that are found on the global security (also called *cell-level* security) panel. You cannot configure a different authentication mechanism or user registry on an individual server basis. This feature is limited to cell-level configuration only. Also, when global security is disabled, you cannot enable application server security.

By default, server security inherits all of the values that are configured for global security (cell-level security). To override the security configuration at the server level, click **Servers > Application Servers >** *server_name*. Under Security, click **Server Security > Additional properties** and click any of the following panels:

- **CSIv2 inbound authentication**
- **CSIv2 inbound transport**
- **CSIv2 outbound authentication**
- **CSIv2 outbound transport**
- **z/SAS authentication**
- **Server-level security**

After modifying the configuration in any of these panels and clicking **OK** or **Apply**, the security configuration for that panel or set of panels now overrides cell-level security. Other panels that are not overridden continue to be inherited at the cell-level. However, you can always revert back to the cell-level configuration at any time. On the Server Security panel, click to revert back to the global security configuration on these panels:

- **Use cell security**
- **Use cell CSI**
- **Use cell SAS**

A number of additional z/SAS attributes that can be considered for security at a server level, such as:
- Local identity
- Remote identity
- Sync to thread allowed

For more information, see “Global security and server security” on page 899.

1. Start the administrative console for the deployment manager. To get to the administrative console, go to http://host.domain:9060/ibm/console. If security is disabled, you can enter any ID. If security is enabled, you must enter a valid user ID and password, which is either the administrative ID (configured for the user registry) or a user ID entered as an administrative user. To add a user ID as an administrative user, click **System Administration > Console settings > Console users**.

2. Configure global security if you have not already done so. Go to the “Configuring global security” on page 881 article for detailed steps. After global security is configured, configure server-level security.
3. To configure server-level security, click **Servers > Application Servers >** *server name*. Under Security, click **Server security**. The status of the security level that is in use for this application server is displayed.

   By default, you can see that global security, CSI, and z/SAS have not been overridden at the server level. CSI and z/SAS are authentication protocols for RMI/IIOP requests. The Server Level Security panel lists attributes that are on the Global Security panel and can be overridden at the server level. Not all of the attributes on the Global Security panel can be overridden at the server level, including Active Authentication Mechanism and Active User Registry.
4. To disable security for this application server, go to the Server Level Security panel, clear the **Enable global security** option and click **OK** or **Apply**. Click **Save**. By modifying the Server Level Security panel, you can see that this flag overrides the cell-level security.
5. To configure CSI at the server level, you can change any panel that starts with CSI. By doing so, all panels that start with CSI will override the CSI settings specified at the cell level. This change includes all authentication and transport panels for CSI. See the “Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols” on page 1161 article for more detailed steps regarding configuring CSI authentication protocol.

Typically server-level security is used to disable user security for a specific application server. However, this can also be used to disable (or enable) the Java 2 Security Manager, and configure the authentication requirements for RMI/IIOP requests both incoming and outgoing from this application server.

After you modify the configuration for a particular application server, you must restart the application server for the changes to become effective. To restart the application server, go to **Servers > Application servers** and click the server name that you recently modified. Then, click the **Stop** button and then the **Start** button.

If you disabled security for the application server, you can typically test a URL that is protected when security is enabled.

Server security settings:

Use this page to configure server security and override the global security settings. If you need to revert to the global security defaults, deselect the appropriate check box in the administrative console.

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

1. Click **Servers** > **Application servers** > *server_name*.
2. Under Security, click **Server security**.

You can disable security on individual application servers while global security is enabled. However, you cannot enable security on an individual application server while global security is disabled. While application server security is disabled for user requests, administrative and naming security is still enabled for that application server so that the administrative and naming infrastructure remains secure. To avoid problems, verify that the naming security has **Everyone** access to the naming function that you use within your user code. You do not need to configure administrative security, because user code does not directly access administrative functions. User code accesses administrative functions through the supported scripting tools.

Server-level security:

Specifies whether the server overrides cell defaults for security.

To revert to the cell defaults for Server-level security, click **Use cell security**. Click **Apply** and then select **Save** to validate the changes at the server level.

Default False

CSI:

Specifies whether the server overrides cell defaults for the CSI protocol.

Default False

SAS:

Specifies whether the server overrides cell defaults for the Secure Authentication Service (SAS) or z/OS Secure Authentication Service (z/SAS) protocol.

Default False

Server-level security settings:

Use this page to enable server level security and specify other server level security configurations.

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

1. Click **Servers > Application Servers >** *server_name*.
2. Under Security, click **Server security**.
3. Under Additional properties, click **Server-level security**.

Enable global security:

Use this flag to disable or enable security again for this application server while global security is enabled. Server security is enabled by default when global security is enabled. You cannot enable security on an application server while global security is disabled. Administrative (administrative console and wsadmin) and naming security remain enabled while global security is enabled, regardless of the status of this flag.

Default Disable

Enforce Java 2 security:

Specifies that the server enforces Java 2 Security permission checking at the server level. When cleared, the Java 2 server-level security manager is not installed and all of the Java 2 Security permission checking is disabled at the server level.

If your application policy file is not set up correctly, see the documentation on configuring an application policy in a `was.policy` file.

Default Disabled

Enforce fine-grained JCA security: Enable this option to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data.

Default Disabled

Use domain qualified user IDs:

Specifies whether user IDs returned by getUserPrincipal()-like calls are qualified with the server level security domain within which they reside.

Default	Disabled

Cache timeout:

Specifies the timeout value for server level security cache in seconds.

Data type	Integer
Units	Seconds
Default	600
Range	Greater than 30 seconds. Avoid setting cache timeout value to 30 seconds or less.

Issue permission warning:

Specifies whether a warning is issued during application installation when an application requires a Java 2 permission that is normally not granted to an application.

WebSphere Application Server provides support for policy file management. A number of policy files are included in WebSphere Application Server. Some of these policy files are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. In dynamic policy files, the code bases are evaluated at run time using configuration data. You can add or remove permissions, as needed, for each code base. However, do not add, remove, or modify the existing code bases. The real code base is dynamically created from the configuration and run-time data. The `filter.policy` file contains a list of permissions that an application does not have, according to the J2EE 1.3 Specification. For more information on permissions, see the documentation on the Java 2 security policy files.

Default	Enabled

Active protocol:

Specifies the active server level security authentication protocol when server level security is enabled.

You can use an Object Management Group (OMG) protocol called Common Secure Interoperability Version 2 (CSIv2) for more vendor interoperability and additional features. If all of the servers in your entire security domain are Version 5.0 servers, it is best to specify **CSI** as your protocol.

Data type	String
Default	CSI and SAS
Range	CSI, CSI and SAS

RACF server class profiles:

The Resource Access Control Facility (RACF) server class profiles are used to control dynamic application environments. Dynamic application environments are displayed and controlled separately from static application environments.

The Resource Access Control Facility (RACF) server class profiles are used to:

1. Permit the unauthorized WebSphere Application Server servant access to controller services
2. Control dynamic application environments, which are displayed and controlled differently from static application environments

You can choose between two SERVER class profiles. You need one of these profiles, and which profile you need correlates to dynamic application environment (DAE) support.

When Dynamic Application Environments are supported, use:

```
RDEFINE SERVER CB.<server>.<cluster>.<cell>  UACC(NONE)
PERMIT <SR_userid> ACC(READ)
```

When Dynamic application environments are not supported (static application environments), use:

```
RDEFINE SERVER CB.<server>.<cluster>    UACC(NONE)
PERMIT < SR_userid> ACC(READ)
```

Note: To use dynamic application environment commands you must be running z/OS Version 1 Release 4 or above with the WLM-DAE support PTF APAR OW54622 enabled.

To set up both the three-part or four-part RACF server class profiles for the application server or cluster for your dynamic application environment, the user ID for the servant must be given read access to both of the profiles.

Three-part profile

The existing three-part profile has the form:

<subsystem_type>.<subsystem_name>.<application_environment_name>

where:

- *<subsystem_type>* is **CB**
- <subsystem_name> is the application server short name.
- *<application_environment_name>* is the application server generic short name, as specified in the WebSphere Application Server variables. If the server resides in a cluster, the name specified here must match the cluster short name. If the server does not reside in a cluster, the name must match the name specified on the ClusterTransitionName custom property for the server .

Four-part profile

The four-part profile adds the cell name to avoid ambiguities with existing profile names. The four-part profile has the form:

<subystem_type>.<subsystem_name>.<application_environment_name>.<cell_name>

where:

- <cell_name> is the short name of the cell containing this application server.

Examples of profile names

Three-part profile names:

- CB.T5SRV1.T5CL1 (the application server with the short name T5SRV1 and generic short name T5CL1)
- CB.*.T5CL1 (all application servers in the generic short name of T5CL1)
- CB.*.* (any application server in the sysplex)

Four-part profile names:

- CB.T5SRV1.T5CL1.T5CELL (the application server with the short name T5SRV1, and generic short name T5CL1 that resides in the cell T5CELL)
- CB.*.T5CL1.T5CELL (all servers in the generic short name of T5CL1 in the T5CELL)

- CB.*.*.T5CELL (any server in the cell named T5CELL)

If you do not want to discriminate between any of the application servers, you can eliminate all the specified profiles and use a generic form to cover the three and four-part names for all the servers in the sysplex:

- CB.*.T5*
- CB.*.T5*.*

Administrative console 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 console

Four administrative roles are defined to provide degrees of authority needed to perform certain WebSphere Application Server administrative functions from either the administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The four administrative security roles are defined in the following table:

administrative roles

Role	Description
monitor	Least privileged where a user can view the WebSphere Application Server configuration and current state.
configurator	Monitor privilege plus the ability to change the WebSphere Application Server configuration.
operator	Monitor privilege plus the ability to change the run-time state, such as starting or stopping services.
administrator	Operator plus configuration privilege and the permission required to access sensitive data including the server password, LTPA password, LTPA, keys, and so on.

When global security is enabled, the administrative subsystem role-based access control is enforced. The administrative subsystem includes security server, user registry, and all the Java Management Extensions (JMX) MBeans. When 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.

WebSphere Application Server for z/OS security customization dialogs prime the administrative subsystem to accept the MVS identities of all started WebSphere system tasks (controllers, servants, and so on) as WebSphere administrators and to accept the configured WebSphere administrator identity.

If a Lightweight Directory Access Protocol (LDAP) or Custom registry is specified, you must ensure that customization provided to facilitate using Local OS is removed. Once an LDAP or Custom registry is used, the configured server identities are used for work executed by the system instead of by the started task identities. You must delete pre-configured WebSphere Configuration Group and Administrator identity from the console group and console users respectively.

SAF authorization for administrative roles

The value of the `com.ibm.security.SAF.authorization` setting controls whether SAF EJBROLE profiles or the console settings are used to control access to administration profiles rather than the console users. With System Authorization Facility (SAF) authorization any values in the console users and console groups are ignored.

WebSphere authorization for administrative roles

If WebSphere authorization (rather than SAF authorization) is used to restrict access to Java 2 Platform, Enterprise Edition (J2EE) roles, WebSphere Application Server for z/OS automatically maps the server identity specified when enabling global security to the administrative role. Also, when global security is enabled, WebSphere Application Servers on z/OS 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. This is why the WebSphere Application Server run-time code, which runs under the server identity, requires authorization to execute run-time 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 Server global security
- Enforce or disable Java 2 Security
- Change the LTPA password or generate keys
- Assign users and groups to administrative roles

A special configuration is not required to enable the server identity (as specified) when enabling global security for administrative use because the server identity is automatically mapped to the administrator role. You can add or remove users and groups to or from the administrative roles from the WebSphere Application Server administrative console. However, a server restart is required for the changes to take effect. A best practice is to map a group, rather than specific users, to administrative roles because it is more flexible and easier to administer. By mapping a group to an administrative 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.

Administrative roles

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 has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if security is not enabled.

When enabling security, you can assign one or more users and groups to administrative 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.

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. They affect the content of the WebSphere Application Server name space. There are generally 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
: Users can query of 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
: Users can perform write operations such as JNDI **bind**, **rebind**, or **unbind**, and CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.

CosNamingCreate
: Users can create new objects in the name space through such operations as JNDI createSubcontext and CosNamingWrite operations. The special subject AllAuthenticated is the default policy for this role.

CosNamingDelete
: Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

When you configure a local OS user registry to use with WebSphere Application Server for z/OS, there are some additional considerations. Refer to Configuring user registries and Steps for selecting a local OS registry for more information. If you specify an LDAP or a custom registry, you must remove local OS customization by deleting the pre-configured WebSphere configuration group and administrator identity from the console group. Then delete the console users.

Additionally, a Server special-subject is assigned to all the four CosNaming roles by default. The Server special-subject provides a WebSphere Application Server server process, which runs under the server identity, access to all the CosNaming operations. Note that the Server special-subject does not display and cannot be modified through the administrative console or other administrative tools.

No special configuration is required to enable the server identity (as specified) when enabling global security for administrative use because the server identity is automatically mapped to the administrator role.

No special configuration is required to enable the server identity (as specified) when enabling global 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. (Note that when SAF Authorization is chosen, no server restart is needed to authorize additional users or groups.) 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. Note that when System Authorization Facility (SAF) authorization is selected, you do not need to restart the server to authorize additional users or groups.

The CosNaming authorization policy is only enforced when global security is enabled. When global 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.

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 run time. Typically, J2EE applications

access the name space and the identity they use is that of the user that authenticated to WebSphere Application Server when they access the J2EE application. Unless the J2EE application provider clearly communicates the expected Naming roles, use caution when changing the default naming authorization policy.

Assigning users to administrator roles

Using System Authorization Facility (SAF) authorization to control access to administrative roles: When `com.ibm.security.SAF.authorization` is set to **true**, SAF EJBROLE profiles are used to control access to administrative roles.

If you selected **Use SAF EJBROLE profiles to enforce Java 2 Platform, Enterprise Edition (J2EE) roles** during security domain setup in the Customization Dialog, then the following administrative roles were defined by the customization jobs. (Note that the security domain name might or might not have been specified during security domain setup, and `configGroup` represents the WebSphere configuration group name that you chose.) Note that SAF role names are case sensitive.

```
RDEFINE EJBROLE (optionalSecurityDomainName.)administrator UACC(NONE)
RDEFINE EJBROLE (optionalSecurityDomainName.)monitor       UACC(NONE)
RDEFINE EJBROLE (optionalSecurityDomainName.)configurator  UACC(NONE)
RDEFINE EJBROLE (optionalSecurityDomainName.)operator      UACC(NONE)

PERMIT (optionalSecurityDomainName.)administrator CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
PERMIT (optionalSecurityDomainName.)monitor       CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
PERMIT (optionalSecurityDomainName.)configurator  CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
PERMIT (optionalSecurityDomainName.)operator      CLASS(EJBROLE) ID(configGroup) ACCESS(READ)
```

If you decide at a future date to turn on SAF authorization, you must issue these Resource Access Control Facility (RACF) commands to enable proper WebSphere Application Server operation. You can give a user access to all administrative functions by connecting it to the configuration group:

```
CONNECT  mvsid  GROUP(configGroup)
```

You can also permit individual users to specific roles issuing the following RACF command:

```
PERMIT (optionalSecurityDomainName.)rolename CLASS(EJBROLE) ID(mvsid) ACCESS(READ)
```

You do not need to restart the server for SAF EJBROLE changes to take effect. However, after the SAF changes have been made, you must issue the following RACF command (or equivalent for your security system) to refresh the in-memory security tables:

```
SETROPTS RACLIST(EJBROLE)  GENERIC
```

Using WebSphere Authorization to control access to administrative roles: When `com.ibm.security.SAF.authorization` is set to **false**, WebSphere Authorization and the administrative console are used to control access to administrative roles.

In the administrative console, click **System Administration > Console settings**. Click either **Console Users** or **Console Groups**.

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 Select from special subject menu, and click **OK**. If no validation error exists, the specified group or special subject displays 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, expand the **filter** folder on the right panel and modify the filter. For example, setting the filter to `user*` only displays users with the `user` prefix.

6. After the modifications are complete, click **Save** to save the mappings.
7. Restart the server for changes to take effect.

The task of assigning users and groups to administrative roles is performed to identify users for performing WebSphere Application Server administrative functions. Administrator roles are used to control access to WebSphere Application Server administrative functions. There are four roles: administrator, configurator, operator and monitor.

Administrator role
Users and groups assigned to the administrator role can perform all administrative operations and can set up both Java 2 Platform, Enterprise Edition (J2EE) role-based and Java 2 security policy.

Configurator role

Users assigned to the configurator role can perform all of the day-to-day configuration tasks including installing and uninstalling applications, assigning users and groups to role mapping for applications, setting run-as configurations, setting up Java 2 security permissions for applications, and customizing Common Secure Interoperability Version 2 (CSIv2), z/OS Security Authentication Service (z/SAS), and Secure Sockets Layer (SSL) configurations.

Operator role
Users assigned to the operator role can view the WebSphere Application Server configuration and its current state, but also can change the run-time state such as stopping and starting services.

Monitor role
Users assigned the monitor state can view the WebSphere Application server configuration and its current state only.

Before you assign users to administrative roles (administrator, configurator, operator, and monitor), you must set up your user registry, which can be Lightweight Directory Access Protocol (LDAP), local OS, or a custom registry. You can set up your user registries without enabling security.

Console groups and CORBA naming service groups:

Use the Console Groups page to give groups specific authority to administer the WebSphere Application Server using 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 groups page to manage CORBA Naming Service groups settings.

To view the Console Groups administrative console page, click **System Administration > Console Groups**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Group (Console groups):

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

Group (CORBA naming service groups):

Identifies CORBA naming service groups.

The ALL_AUTHENTICATED group has the following role privileges: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The EVERYONE group indicates that the users in this group have CosNamingRead privileges only.

Data type: String
Range: ALL_AUTHENTICATED, EVERYONE

Role (Console group):

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain WebSphere Application Server administrative functions:

Administrator
The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Configurator
The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

Operator
The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Monitor
The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

Data type: String
Range: Administrator, Configurator, Operator, and Monitor

Role (CORBA naming service users):

Identifies naming service group roles.

A number of naming roles are defined to provide degrees of authority needed to perform certain WebSphere 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 names of the four roles are the same with WebSphere Advanced Edition, Version 4.0.2. However, the roles now have authority levels from low to high:

Cos Naming Read
Users can query the WebSphere name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The special-subject EVERYONE is the default policy for this role.

Cos Naming Write
Users can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.

Cos Naming Create
Users can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.

Cos Naming Delete
Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.

Data type:	String
Range:	CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete

Assigning users to naming roles

The following steps are needed to assign users to naming roles. In the administrative console, expand **Environment > Naming**, and click **CORBA Naming Service Users** or **CORBA Naming Service Groups**.

1. Click **Add** on the **CORBA Naming Service Users** or **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 either **EVERYONE** or **ALL AUTHENTICATED**. 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 run time.

Special considerations for controlling access to naming roles using SAF authorization:
Considerations for assigning users to naming roles: You can use either System Authorization Facility (SAF) authorization (EJBROLE profiles) or WebSphere Authorization to control access to naming roles. The user registry custom variable `com.ibm.security.SAF.authorization` determines when SAF authorization or WebSphere Authorization is used. For a discussion of the CosNaming roles, see Administrative console and naming service authorization. You can also refer to Assigning users to naming roles.

Using SAF authorization to control access to naming roles: When com.ibm.security.SAF.authorization is set to true, SAF EJBROLE profiles are used to control access to CosNaming functions. If you selected Use SAF EJBROLE profiles to enforce J2EE roles during security domain setup in the Customization Dialog, then the following CosNaming roles were defined by the customization jobs:

```
RDEFINE EJBROLE (optionalSecurityDomainName.)CosNamingRead UACC(READ)
PERMIT (optionalSecurityDomainName.)CosNamingRead  CLASS(EJBROLE)  ID(WSGUEST) ACCESS(READ)
RDEFINE EJBROLE (optionalSecurityDomainName.)CosNamingWrite  UACC(READ)
RDEFINE EJBROLE (optionalSecurityDomainName.)CosNamingCreate UACC(READ)
RDEFINE EJBROLE (optionalSecurityDomainName.)CosNamingDelete UACC(READ)
```

If you decide at a future date to turn on SAF authorization, you must issue these RACF commands to enable proper WebSphere Application Server operation. (Change the value WSGUEST is you have chosen a different unauthenticated user ID.)

The default access granted by the customization dialog permits all authenticated users to update the name space. This type of authorizations might be a broader level of authority than you want to provide. Minimally, you must enable the WebSphere Configuration group (servers and administrators) to have READ access to all profiles and permit all WebSphere Application Server for z/OS clients to have READ access to the CosNamingRead profile.

If additional users require access to CosNaming roles, you can permit a user to any of the previous roles as indicated by issuing the following RACF command: `PERMIT (optionalSecurityDomainName.)rolename CLASS(EJBROLE) ID(mvsid) ACCESS(READ)`

Using WebSphere Authorization to control access to naming roles: When `com.ibm.security.SAF.authorization` is set to **false**, WebSphere authorization and the administrative console are used to control access to CosNaming functions.

For information on assigning users to naming roles, refer to Assigning users to naming roles.

Console users settings and CORBA naming service user settings:

Use the Console users settings page to give users specific authority to administer WebSphere Application Server using 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, click **System Administration > Console Users**.

To view the CORBA naming service users administrative console page, click **Environment > Naming > CORBA Naming Service users**.

User (Console users):

Specifies users.

The users entered must exist in the configured active user registry.

Data type: String

User (CORBA naming service users):

Specifies CORBA naming service users.

The users entered must exist in the configured active user registry.

Data type: String

Role (Console users):

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain WebSphere Application Server administrative functions:

Administrator
The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Configurator
The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

Operator
The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Monitor
The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

Data type:	String
Range:	Administrator, Configurator, Operator, and Monitor

Role (CORBA naming service users):

Specifies naming service user roles.

A number of naming roles are defined to provide degrees of authority needed to perform certain WebSphere 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 names of the four roles are the same with WebSphere Application Server, Advanced Edition Version 4.0.2. However, the roles now have authority levels from low to high:

CosNamingRead
Users can query the WebSphere name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The special-subject EVERYONE is the default policy for this role.

CosNamingWrite
Users can perform write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

CosNamingCreate
Users can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

CosNamingDelete
Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

Data type:	String
Range:	CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete

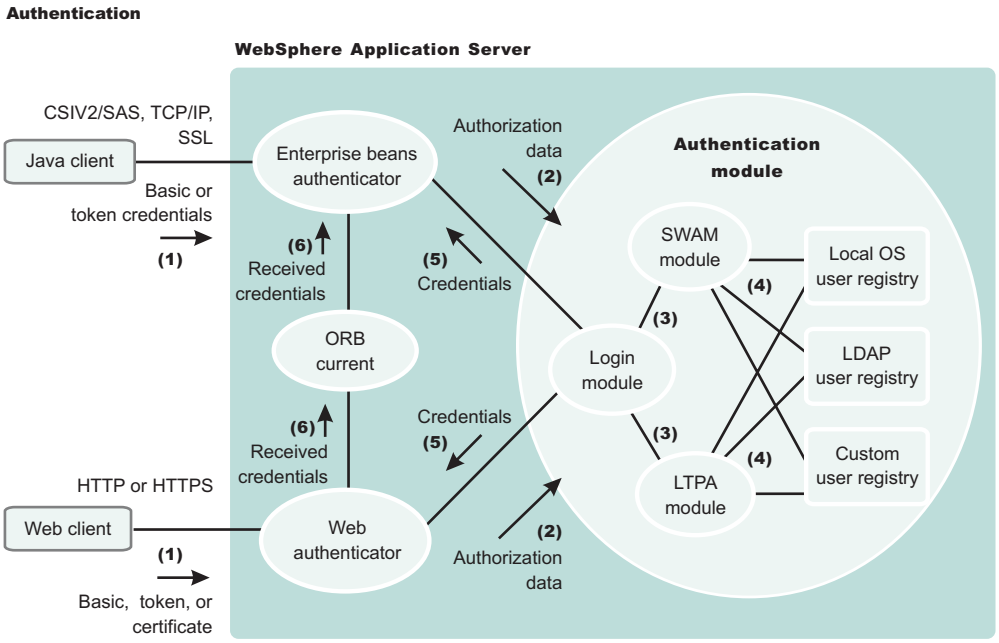
Authentication mechanisms

An *authentication mechanism* defines rules about security information (for example, 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.

Although this product provides multiple authentication mechanisms, you can configure only a single *active* authentication mechanism at a time. The active authentication mechanism is selected when configuring WebSphere Application Server global security.



Authentication Process

The figure demonstrates the authentication process. Basically, authentication is required for enterprise bean clients and Web clients when they access protected resources. Enterprise bean clients (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:

- Common Secure Interoperability Version 2 (CSIv2)
- z/OS Secure Authentication Service (z/SAS)

Web clients use the HTTP or HTTPS protocol to send the authentication information as shown in the previous figure.

The authentication information can be BasicAuth (user ID and password), credential token (in case of Lightweight Third Party Authentication (LTPA) on the z/OS platform), or client certificate. The Web authentication is performed by the Web Authentication module and the enterprise bean authentication is performed by the Enterprise JavaBean (EJB) authentication module, which resides in the CSIv2 and SAS layer or z/SAS layer on the z/OS platform.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:

- Lightweight Third Party Authentication (LTPA). LTPA is the only authentication mechanism supported by WebSphere Application Server Network Deployment.
- Simple WebSphere Authentication Mechanism (SWAM)

The authentication module uses the registry that is configured on the system to perform the authentication (4). Three types of registries are supported: Local OS, Lightweight Directory Access Protocol (LDAP), and custom registry. External registry implementation following the registry interface specified by IBM can replace either the Local OS or the LDAP user registry.

The login module creates a JAAS subject after authentication and stores the credential derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or enterprise beans authenticator (5).

The Web authenticator and the EJB authenticator store the received credentials for the authorization service to use in performing further access control checks.

Steps for selecting an authentication mechanism

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. Implementation is provided to support multiple operating system or operating environment-based user registries such as z/OS System Authorization Facility (SAF) registry and most of the major Lightweight Directory Access Protocol (LDAP)-based user registries. You can use the custom LDAP feature to support any LDAP server by setting up the correct configuration (user and group filters). However, support is not extended to these custom LDAP servers since there are many possibilities that cannot be tested.

The next step in setting up security is to select an authentication mechanism. An authentication mechanism defines rules about security information (for example, 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 valid 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 accounts 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 successfully authenticated client user. Not all credentials are created equal. The abilities of the credential are determined by the configured authentication mechanism.

Although this product provides several authentication mechanisms, only a single active authentication mechanism can be configured at once. The active authentication mechanism is selected when configuring WebSphere global security. WebSphere Application Server for z/OS Version 6.0.x supports the following authentication mechanisms:

- Simple WebSphere Authentication Mechanism (SWAM)
- Light-Weight Third Party Authentication (LTPA)

Note: In future releases, IBM intends to deprecate the ICSF authentication mechanism. It is recommended that you migrate to LTPA. For more information on LTPA, see Lightweight Third Party Authentication.

Steps for selecting the SWAM authentication mechanism:

If you are using Simple WebSphere Authentication Mechanism (SWAM), there is no setup needed as this is the default mechanism.

Note: SWAM is only valid in a base installation. It is not supported in ND.

Steps for selecting LTPA as the authentication mechanism:

You need to start the Administrative Console by specifying URL:
`http://server_hostname:9060/ibm/console`.

Perform the following steps to select Lightweight Third Party Authentication (LTPA) as the authentication mechanism for this server.

1. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**.
2. Enter the password and confirm it in the password fields. This password is used to encrypt and decrypt the LTPA keys during export and import of the keys. Remember this password because you enter it again when the keys from this cell are exported to another cell.
3. Enter a positive integer value in the 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 is prompted to log in. An optimal value for this field depends on your configuration; there is no recommended time. The default value is 120 minutes.
4. Click **Apply** or **OK**. The LTPA configuration is now set.
5. Complete the information in the Global Security panel and press OK. When **OK** or **Apply** is clicked in the Global Security panel the LTPA keys are generated automatically the first time, and therefore, you should not generate the keys manually.

Steps for selecting ICSF as the authentication mechanism:

Integrated Cryptographic Services Facility (ICSF) requires the Cryptographic Coprocessor features of the zSeries processor to be enabled and active. You must have ICSF configured and running on your processor before selecting ICSF as your authentication mechanism.

Note: In future releases, IBM intends to deprecate the ICSF authentication mechanism. It is recommended that you migrate to LTPA. For more information on LTPA, see “Lightweight Third Party Authentication” on page 919.

You need to start the Administrative Console by specifying URL:
`http://server_hostname:9060/ibm/console`.

Perform the following steps to select ICSF as the authenication mechanism for this server.

1. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > ICSF**.
2. In the **Encryption cryptographic key** field, specify the label of the cryptographic key to use for single signon (SSO) tokens for Web applications and administrative security when using the Simple Object Access Protocol (SOAP) HTTP connector.
3. Enter a positive integer value in the Timeout field. Specifies the time period in which an ICSF token expires. Verify that this time period is longer than the cache time-out that is configured in the Global Security panel.
4. Click **Apply** or **OK**. The ICSF configuration is now set.

Configuring authentication mechanisms

Configure authentication mechanisms by clicking **Authentication Mechanisms** under **Security > Global security** in the administrative console.

- If you are using Simple WebSphere Authentication Mechanism (SWAM), no setup is needed. Follow the instructions in “Configuring Lightweight Third Party Authentication” on page 920 to set up Lightweight Third Party Authentication (LTPA).

- For LTPA, follow the steps in “Configuring single signon” on page 931 for most situations. If trust association is required, follow the steps in “Configuring trust association interceptors” on page 928.

Simple WebSphere authentication mechanism:

The Simple WebSphere authentication mechanism (SWAM) is intended for simple, non-distributed, single application server run-time environments. 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 signon (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.

Lightweight Third Party Authentication:

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. It supports forwardable credentials and single signon (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 signon (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.

For the Lightweight Directory Access Protocol (LDAP), the realm name is the host:port of the LDAP server.

The LTPA protocol uses cryptographic keys (LTPA keys) to encrypt and decrypt user data that passes between the servers. These keys need to 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 participating 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).

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 it has not expired and 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 (cell, 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, Network Deployment supports both the LTPA and the Integrated Cryptographic Services Facility (ICSF) protocols.

When security is enabled for the first time with LTPA, configuring LTPA is normally the initial step performed.

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	LocalOS User Registry	LDAP User Registry	Custom User Registry
LTPA	Yes	Yes	Yes	Yes	Yes
ICSF	Yes	Yes	Yes	Yes	Yes

Configuring Lightweight Third Party Authentication:

The following steps are needed to configure Lightweight Third Party Authentication (LTPA) when setting up security for the first time:

1. Access the administrative console by typing `http://localhost: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 > Global security**.
3. Under Authentication, click **Authentication mechanisms > LTPA**.
4. Enter the password and confirm it in the password fields. This password is used to encrypt and decrypt the LTPA keys during export and import of the keys. Remember this password because you enter it again when the keys from this cell are exported to another cell.
5. Enter a positive integer value in the **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 is prompted to log in.

   When the token expires, the request is rejected and the user must log in again.

   An optimal value for this field depends on your configuration. The default value is 30 minutes.
6. **Optional:** In the **Key file name** field, specify the name of the file that is used when you import or export keys. You can use this field in conjunction with the **Import keys** and **Export keys** buttons at the top of the panel.
7. 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 required to enable security, starting with single signon (SSO) (if SSO is required).
8. Complete the information in the Global Security panel and click **OK**. The LTPA keys are generated automatically the first time. Do not generate the keys manually.

The previous steps configure LTPA by setting passwords that generate LTPA keys.

After configuring LTPA, complete the following steps to work with your key files:
1. Generate key files.
2. Export key files.
3. Import key files.

4. If you are enabling security, make sure that you complete the remaining steps starting with enabling SSO.
5. If you generated a new set of keys or imported a new set of keys, verify that the keys are saved 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 product servers that are participating in the protection domain. If the clock skew is too high between servers, the LTPA token appears prematurely expired and causes authentication or validation failures.

Configuring Lightweight Third Party Authentication keys:

Generating keys:

Lightweight Third Party Authentication (LTPA) keys are automatically generated when a password change is detected. The first time that you set the LTPA password, as part of enabling security, the LTPA keys are automatically generated after **OK** or **Apply** is clicked in the LTPA panel. You do not have to click **Generate Keys** in this situation. Complete the following steps in the administrative console to generate a new set of LTPA keys:

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Verify that all the WebSphere Application Server processes are running (cell, 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 to bring them back up.
3. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**.
4. Click **Generate Keys** if you want to use the existing password. This action generates a new set of keys that are encrypted with the same password as the old set of keys. Regardless of the password change, a new set of keys is generated when you click **Generate Keys**. This new set of keys is not propagated to the run time unless saved; save the files immediately.
5. Enter the new password and confirm it, to use a new password to generate keys. Click **OK** or **Apply**. A new set of keys is generated. A message indicating that a new set of keys is generated displays on the console. Do not click **Generate Keys**. These new keys are propagated to the run time after you save them.
6. Click **Save** to save the keys. After a new set of keys is generated and saved, the generated keys are not used in the configuration until the WebSphere Application Server is restarted. In a Deployment Manager environment the node agents and application servers must also be recycled to accept the new keys. If any of the node agents are down, run a manual file synchronization utility from the node agent machine to synchronize the security configuration from the deployment manager. The next sections describe the process of exporting and importing the keys.

Exporting keys:

To support single signon (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, share the LTPA keys and the password among the domains. Make sure that the time on the domains is similar to prevent the tokens from appearing as expired between the cells. You can use **Export Keys** to export the LTPA keys to other domains or cells. Complete the following steps in the administrative console to export key files for LTPA:

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**.
3. In the **Key file name** field, enter the full path of a file for key storage. This file needs write permissions.
4. Click **Export Keys**. A file is created with the LTPA keys. Exporting keys fails if a new set of keys is generated or imported and not saved prior to exporting. To avoid failure, make sure that you save the new set of keys (if any) prior to exporting them.
5. Click **Save** to save the configuration.

Importing keys:

To support SSO in WebSphere Application Server across multiple WebSphere Application Server domains or cells, share the LTPA keys and the password among the domains. You can use **Import Keys** to import the LTPA keys from other domains. Verify that key files are exported from one of the cells involved, into a file. Complete the following steps in the administrative console to import key files for LTPA.

After a new set of keys is generated and saved, the generated keys are not used in the configuration until the WebSphere Application Server is restarted. In a Deployment Manager environment, the node agents and application servers must also be recycled to accept the new keys. If any of the node agents are down, run a manual file synchronization utility from the node agent machine to synchronize the security configuration from the deployment manager.

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**.
3. Change the password in the **password** fields to match the password in the cell from which you are importing the keys.
4. Click **Save** to save the new set of keys in the repository. This step is important to complete before importing the keys. If the password and the keys do not match, the servers fail. If the servers fail, turn off security and redo these steps.
5. In the **Key file name** field, enter the full path of a file for key storage. This file needs read permissions.
6. Click **Import Keys**. The keys are now imported into the system.
7. Click **Save** to save the new set of keys in the repository. It is important to save the new set of keys to match the new password so that no problems are encountered starting the servers later.

Lightweight Third Party Authentication settings:

Use this page to configure Lightweight Third Party Authentication (LTPA) settings.

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

1. Click **Security > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**.

If you are configuring security for the first time, only the password is required. After the password is entered, click **Apply**. Under Additional Properties, click **Single signon (SSO)** and enter the domain name. Make sure that SSO is enabled. Click **Apply**. In the Global security panel under **Security > Global security**, click **Custom Properties**. A list of security properties is displayed. Click the **control_region_security_enable_trusted_applications** property. On the new window, change the **Value** field from `false` to `true`, and click **Apply**.

To complete the security setup, make sure that the appropriate registry is set up and click **Apply** from the Global security panel. When security is enabled and any of these properties change, go to the Global security panel under **Security > Global security** and click **Apply** to validate the changes.

Generate Keys:

Specifies whether the server generates new Lightweight Third Party Authentication (LTPA) keys.

When security is turned on for the first time with LTPA as the authentication mechanism, the LTPA keys are automatically generated with the password entered in the panel. If you need a new set of keys to generate using the previously set password, click **Generate Keys**. If a new password is used, do not click this option. After the new password is entered and **OK** or **Apply** is clicked, a new set of keys is generated. *A new set of generated keys is not used until you save them.*

Import Keys:

Specifies whether the server imports new LTPA keys.

To support single signon (SSO) in the WebSphere product across multiple WebSphere 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, enter the appropriate password, click **OK** and click **Save**. Then, enter the directory location where the LTPA keys are located prior to clicking **Import keys**. Do not click **OK** or **Apply**, but save the settings.

Export Keys:

Specifies whether the server exports LTPA keys.

To support single signon (SSO) in the WebSphere product across multiple WebSphere 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 **Key file name** field and click **Export Keys**. The encrypted keys are stored in the specified file.

Password:

Specifies the password to encrypt and decrypt the LTPA keys. Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for a Lotus Domino server.

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 used to encrypt and decrypt the LTPA keys.

Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for a Lotus Domino server.

Data type String

Timeout:

Specifies the time period in minutes at which an LTPA token expires. Verify that this time period is longer than the cache timeout configured in the Global security panel.

Data type Integer
Units Minutes
Default 120

Key file name:

Specifies the name of the file used when importing or exporting keys.

Enter a fully qualified key file name, and click **Import Keys** or **Export Keys**.

Data type	String

Integrated Cryptographic Services Facility settings:

Use this page to configure Integrated Cryptographic Services Facility (ICSF) settings.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Authentication mechanisms** > **ICSF**.

Timeout:

Specifies the time period in which an ICSF token expires. Verify that this time period is longer than the cache timeout that is configured in the Global Ssecurity panel.

Data type	Integer
Units	Minutes
Default	120

Encryption cryptographic key:

Specifies the label of the cryptographic key to use for single signon tokens for Web applications and administrative security when using the Simple Object Access Protocol (SOAP) HTTP connector.

You can create the cryptographic key in a Cryptographic Key Data Set (CKDS) accessible by ICSF. For additional information, see the *z/OS Integrated Cryptographic Services Overview* manual or the *OS/390 Integrated Cryptographic Services Overview* manual

Data type	String

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 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.

The demand is growing to provide customers with a trust association solution between IBM WebSphere Application Server and other Web authentication servers that act as a reverse proxy security server (IBM Tivoli Access Manager for e-business - WebSEAL, Caching Proxy) as an entry point to all service requests (See the first figure). This implementation design intends to have the proxy server as the only exposed entry point. The proxy server authenticates all requests that come in and provides coarse, granularity junction point authorization.

In this setup, the 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 the 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

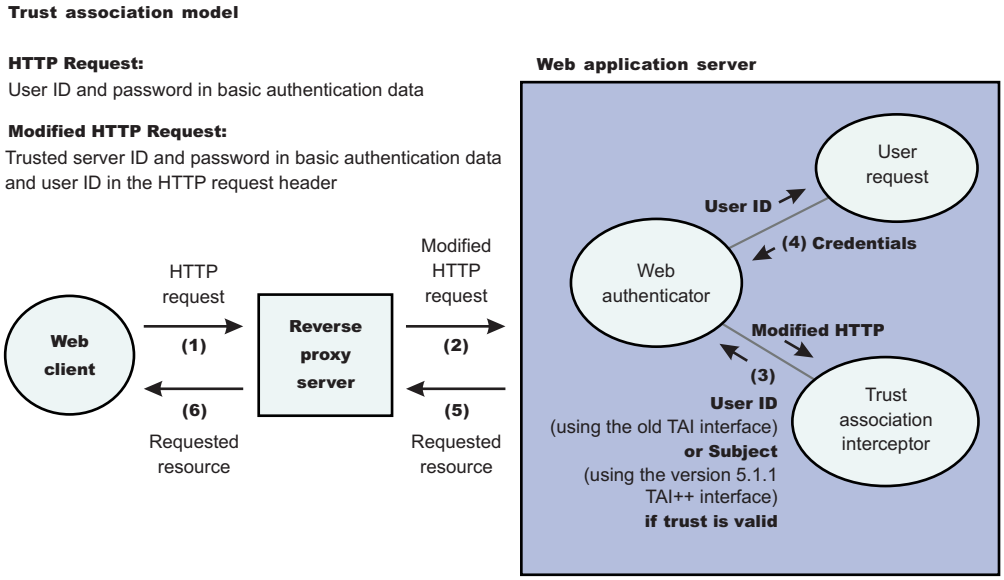
The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests received from a reverse proxy server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full 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 WebSphere Application Serve TAI interface is provided to support WebSEAL Version 4.1. If you plan to use WebSEAL 5.1, 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, but does not support WebSEAL Version 4.1. For an explanation of security attribute propagation, see “Security attribute propagation” on page 1052.



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. See Figure 2. From a WebSEAL management

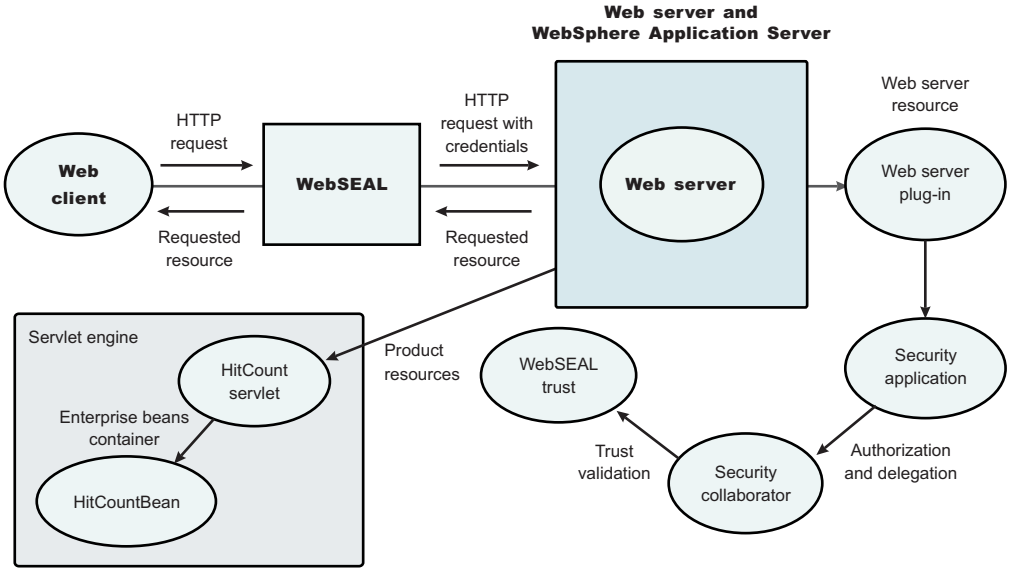
perspective, a junction is created with WebSEAL on one end, and the product Web server on the other end. A junction is a logical connection created to establish a path from the WebSEAL server to another server.

In this setup, a request for Web resources 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, the 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, the 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 it to the client user.

WebSEAL server

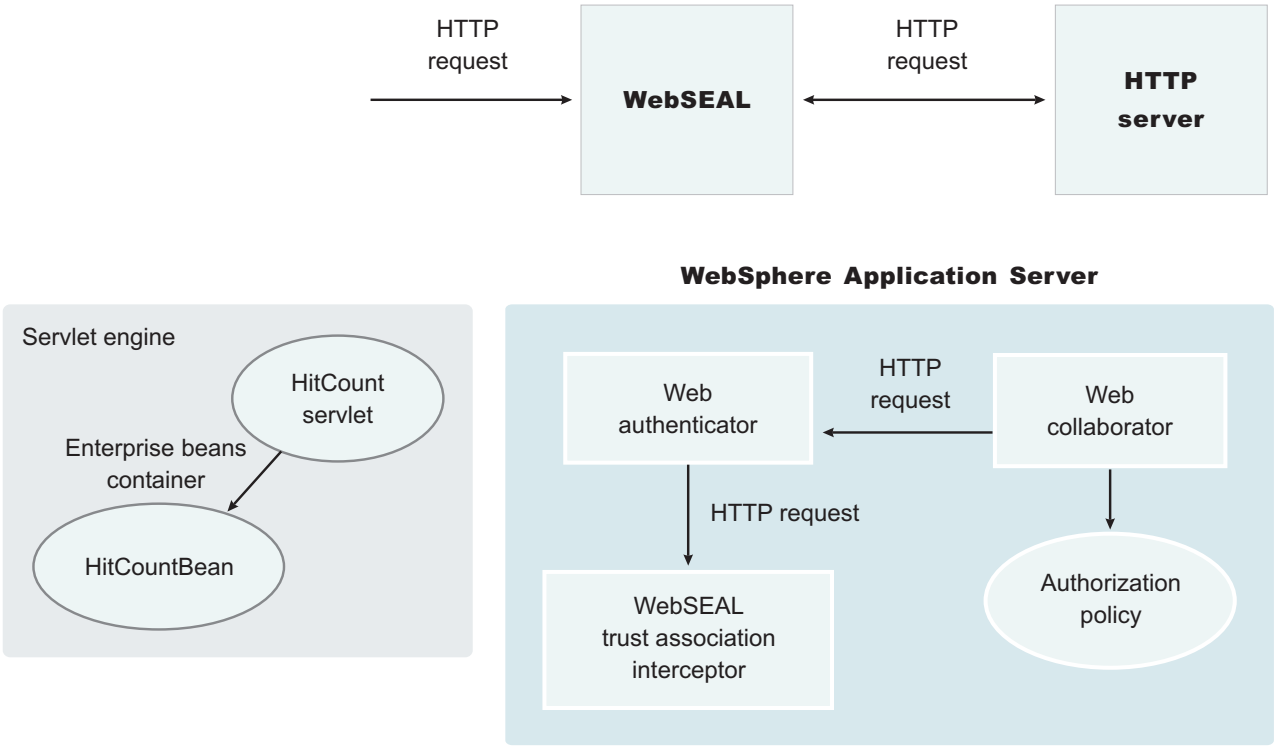
The policy director delegates all of the Web requests to its Web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server consults a Lightweight Directory Access Protocol (LDAP) directory. It can also map the original user ID to another user ID, such as when global single signon (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 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. It 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. Although the existing interface is still supported, it is being deprecated in a future release.

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 supports the `com.ibm.wsspi.security.tai.TrustAssociationInterceptor` interface

For more information, see Trust association interceptor support for Subject creation .

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 the 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, legacy 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 a 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.

Configuring trust association interceptors:

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 by typing `http://localhost: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 > Global security**.
3. Under Authentication, click **Authentication mechanisms > LTPA**.
4. Under Additional properties, click **Trust association**.
5. Select the **Enable trust association** option.
6. Under Additional properties, click **Interceptors**. The default value appears.
7. Verify that the appropriate trust association interceptors are listed. If you need to use a WebSEAL trust association interceptor, see “Configuring single signon using the trust association interceptor” on page 940 or “Configuring single signon using trust association interceptor ++” on page 941. 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:

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 passed by the proxy server. Use this page to configure trust association settings.

To view this administrative console page, complete the following steps:
1. Click **Security > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**.
3. Under Additional properties, click **Trust association**.

When security is enabled and any of these properties change, go to the **Global security** 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 > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**.

3. Under Additional Properties, click **Trust association > Interceptors**.

When security is enabled and any of these properties are changed, go to the Global security 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`

Single signon:

With single signon (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.

Web users can authenticate once to a WebSphere Application Server or to a Domino server. Without logging in again, Web users can access any other WebSphere Application Servers or Domino servers in the same Domain Name Service (DNS) domain that are enabled for SSO. This authentication is accomplished by configuring the WebSphere Application Servers and the Domino servers to share authentication information.

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 single signon 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. 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 user registry. Domino servers do not support custom registries, but you can use a Domino-supported registry as a custom registry within WebSphere Application Server. For more information on custom registries, see *Introduction to custom registries*.

  You can use a Domino directory (configured for LDAP access) or other LDAP directory for the user registry. The LDAP directory product must have WebSphere Application Server support. Supported products include both Domino and IBM SecureWay LDAP directory servers. Regardless of the choice to use an LDAP or a 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 then 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 5.0.6a for iSeries 400 or later and Domino Release 5.0.5 or later for 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 is supported.
  - You can use any HTTP Web server 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.

Configuring single signon:

With single signon (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.

SSO is supported 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.

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 signon is a subset of LTPA, it is recommended that you read “Lightweight Third Party Authentication” on page 919 for more information.

When you enable security attribute propagation, the following cookies are added to the response:

LtpaToken
The LtpaToken is used for interoperating 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 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 1052.

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 located in the SSO configuration panel in the administrative console. To access this panel, complete the following steps: 1. Click **Security > Global security**. 2. Under Authentication, click **Authentication mechanisms > LTPA**. 3. Under Additional properties, click **Single signon (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 allows for multiple attributes 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 > Global security**. 2. Under Authentication, click **Authentication mechanisms > LTPA**. 3. Under Additional properties, click **Single signon (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 > Global security**. 2. Under Authentication, click **Authentication mechanisms > LTPA**. 3. Under Additional properties, click **Single signon (SSO)**.

The following steps are required to configure SSO for the first time.

1. Access the administrative console by typing `http://localhost:`*`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 > Global security** .
3. Under Authentication, click **Authentication mechanisms > LTPA**.
4. Under Additional properties, click **Single signon (SSO)**.
5. Click the **Enabled** option if SSO is disabled. After you click **Enabled**, make sure that you complete the remaining steps to enable security.
6. Click the **Requires SSL** option if all of the requests are expected to use HTTPS.

7. Enter the fully-qualified domain names in the **Domain name** field where SSO is effective. The cookie is sent for all of the servers that are contained within the domains that you specify in this field. 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.

   You can configure the **Domain name** field using any of the following values:

Domain name value type	Example
Blank	
Single domain name	austin.ibm.com
UseDomainFromURL	UseDomainFromURL
Multiple domain names	austin.ibm.com;raleigh.ibm.com
Multiple domain names and UseDomainFromURL	• austin.ibm.com;raleigh.ibm.com • UseDomainFromURL

   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.

   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.

   For more information, see “Single signon settings” on page 934.

8. **Optional:** Enable the **Interoperability mode** option if you want to allow 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 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 are added to the response. Otherwise, only the LtpaToken2 is added to the response. If the **Web inbound security attribute propagation** option is disabled, then only the LtpaToken is added to the response.
9. **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 1052.

   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 using WSSecurityHelper application programming interfaces (APIs).

If you find 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.

10. Click **OK**.

For the changes to take effect, save, stop, and restart all the product servers (deployment managers, nodes and Application Servers).

Single signon settings:

Use this page to set the configuration values for single signon (SSO).

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

1. Click **Security > Global Security**.
2. Under Authentication mechanisms, click **LTPA**.
3. Under Additional properties, click **Single signon (SSO)**.

Enabled:

Specifies that the single signon function is enabled.

Web applications that use J2EE FormLogin style login pages (such as the WebSphere Application Server administrative console) require single signon (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 signon 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 signon hosts.

WebSphere 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 signon 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,

WebSphere 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 WebSphere Application Server does not set a domain for the LtpaToken cookie.

If you specify `UseDomainFromURL`, WebSphere Application Server sets the SSO domain name value to the domain of the host used in the URL. 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 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 signon (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 signon (SSO) token is used to log in and recreate the Subject from the user registry. If you disable this option, the Web inbound login module functions the same as it did in previous releases.

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.

Troubleshooting single signon configurations:

This article describes common problems in configuring single signon (SSO) between a WebSphere Application Server and a Domino server and suggests possible solutions.

- Failure to save the Domino Web SSO configuration document

  The client must find Domino server documents for the participating SSO Domino servers. The Web SSO configuration document is encrypted for the servers that you specify. The home server that is indicated by the client location record must point to a server in the Domino domain where the participating servers reside. This pointer ensures that lookups can find the public keys of the servers.

  If you receive a message stating that one or more of the participating Domino servers cannot be found, then those servers cannot decrypt the Web SSO configuration document or perform SSO.

  When the Web SSO configuration document is saved, the status bar indicates how many public keys are used to encrypt the document by finding the listed servers, authors, and administrators in the document.
- Failure of the Domino server console to load the Web SSO configuration document at Domino HTTP server startup

  During configuration of SSO, the server document is configured for **Multi-Server** in the **Session Authentication** field. The Domino HTTP server tries to find and load a Web SSO configuration document during startup. The Domino server console reports the following information if a valid document is found and decrypted: `HTTP: Successfully loaded Web SSO Configuration`.

If a server cannot load the Web SSO configuration document, SSO does not work. In this case, a server reports the following message: `HTTP: Error Loading Web SSO configuration`. Reverting to single-server session authentication.

Verify that only one Web SSO Configuration document is in the Web Configurations view of the Domino directory and in the $WebSSOConfigs hidden view. You cannot create more than one document, but you can insert additional documents during replication.

If you can verify only one Web SSO Configuration document, consider another condition. When the public key of the Server document does not match the public key in the ID file, this same error message can display. In this case, attempts to decrypt the Web SSO configuration document fail and the error message is generated.

This situation can occur when the ID file is created multiple times but the Server document is not updated correctly. Usually, an error message is displayed on the Domino server console stating that the public key does not match the server ID. If this situation occurs, then SSO does not work because the document is encrypted with a public key for which the server does not possess the corresponding private key.

To correct a key-mismatch problem:
1. Copy the public key from the server ID file and paste it into the Server document.
2. Create the Web SSO configuration document again.

- Authentication fails when accessing a protected resource.

  If a Web user is repeatedly prompted for a user ID and password, SSO is not working because either the Domino or the WebSphere Application Server security server cannot authenticate the user with the Lightweight Directory Access Protocol (LDAP) server. Check the following possibilities:
  - Verify that the LDAP server is accessible from the Domino server machine. Use the **TCP/IP ping** utility to check TCP/IP connectivity and to verify that the host machine is running.
  - Verify that the LDAP user is defined in the LDAP directory. Use the **ldapsearch** utility to confirm that the user ID exists and that the password is correct. For example, you can run the following command, entered as a single line:

    ```
    % ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US" -w mypassword
    -h myhost.mycompany.com -p 389
    -b "ou=Rochester, o=IBM, c=US" (objectclass=*)
    ```

    (The percent character (%) indicates the prompt and is not part of the command.) A list of directory entries is expected. Possible error conditions and causes are contained in the following list:
    - No such object: This error indicates that the directory entry referenced by either the user's distinguished name (DN) value, which is specified after the -D option, or the base DN value, which is specified after the -b option, does not exist.
    - Invalid credentials: This error indicates that the password is invalid.
    - Cannot contact the LDAP server: This error indicates that the host name or port specified for the server is invalid or that the LDAP server is not running.
    - An empty list means that the base directory specified by the -b option does not contain any directory entries.
  - If you are using the user's short name (or user ID) instead of the distinguished name, verify that the directory entry is configured with the short name. For a Domino directory, verify the **Short name/UserID** field of the Person document. For other LDAP directories, verify the userid property of the directory entry.
  - If Domino authentication fails when using an LDAP directory other than a Domino directory, verify the configuration settings of the LDAP server in the Directory assistance document in the Directory assistance database. Also verify that the Server document refers to the correct Directory assistance document. The following LDAP values specified in the Directory Assistance document must match the values specified for the user registry in the WebSphere administrative domain:
    - Domain name
    - LDAP host name
    - LDAP port
    - Base DN

Additionally, the rules defined in the Directory assistance document must refer to the base distinguished name (DN) of the directory containing the directory entries of the users.

You can trace Domino server requests to the LDAP server by adding the following line to the server `notes.ini` file:

```
webauth_verbose_trace=1
```

After restarting the Domino server, trace messages are displayed in the Domino server console as Web users attempt to authenticate to the Domino server.

- Authorization failure when accessing a protected resource.

  After authenticating successfully, if an authorization error message is displayed, security is not configured correctly. Check the following possibilities:
  - For Domino databases, verify that the user is defined in the access-control settings for the database. Refer to the Domino Administrative documentation for the correct way to specify the user's DN. For example, for the `DN cn=John Doe, ou=Rochester, o=IBM, c=US`, the value on the access-control list must be set as `John Doe/Rochester/IBM/US`.
  - For resources protected by WebSphere Application Server, verify that the security permissions are set correctly.
    - If granting permissions to selected groups, make sure that the user attempting to access the resource is a member of the group. For example, you can verify the members of the groups by using the following Web site to display the directory contents:
      `Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub`
    - If you have changed the LDAP configuration information (host, port, and base DN) in a WebSphere Application Server administrative domain since the permissions were set, the existing permissions are probably invalid and need to be recreated.
- SSO failure when accessing protected resources.

  If a Web user is prompted to authenticate with each resource, SSO is not configured correctly. Check the following possibilities:
  1. Configure both the WebSphere Application Server and the Domino server to use the same LDAP directory. The HTTP cookie used for SSO stores the full DN of the user, for example, `cn=John Doe, ou=Rochester, o=IBM, c=US`, and the domain name service (DNS) domain.
  2. Define Web users by hierarchical names if the Domino Directory is used. For example, update the **User name** field in the Person document to include names of this format as the first value: `John Doe/Rochester/IBM/US`.
  3. Specify the full DNS server name, not just the host name or TCP/IP address for Web sites issued to Domino servers and WebSphere Application Servers configured for SSO. For browsers to send cookies to a group of servers, the DNS domain must be included in the cookie, and the DNS domain in the cookie must match the Web address. (This requirement is why you cannot use cookies across TCP/IP domains.)
  4. Configure both Domino and the WebSphere Application Server to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. The DNS domain value is found on the Configure Global Security Settings panel of the WebSphere Application Server administrative console and in the Web SSO Configuration document of a Domino server. If you make a change to the Domino Web SSO Configuration document, replicate the modified document to all of the Domino servers participating in SSO.
  5. Verify that the clustered Domino servers have the host name populated with the full DNS server name in the Server document. By using the full DNS server name, Domino Internet Cluster Manager (ICM) can redirect to cluster members using SSO. If this field is not populated, by default, ICM redirects Web addresses to clustered Web servers by using the host name of the server only. It cannot send the SSO cookie because the DNS domain is not included in the Web address. To correct the problem:
     a. Edit the Server document.
     b. Click **Internet Protocols > HTTP** tab.
     c. Enter the full DNS name of the server in the **Host names** field.
  6. If a port value for an LDAP server was specified for a WebSphere Application Server administrative domain, edit the Domino Web SSO configuration document and insert a backslash character (\) into

the value of the **LDAP Realm** field before the colon character (:). For example, replace `myhost.mycompany.com:389` with `myhost.mycompany.com\:389`.

Single signon using WebSEAL or the Tivoli Access Manager plug-in for Web servers:

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 signon (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 signon are available, the TAI interface and the new TAI interface, so named as both use WebSphere Application Server trust association interceptors (TAIs). With TAI, the end-user name is extracted from the HTTP header and forwarded to embedded Tivoli Access Manager where it is used to construct the client credential information and authorize the user. The difference with the new TAI interface is that all user credential information is available in the HTTP header (not just user name). The new TAI is the more efficient of the two solutions as an Lightweight Directory Access Protocol (LDAP) call is not required as it is with TAI. TAI functionality is retained for backwards compatibility.

The following tasks need to be completed to enable single signon 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. “Creating a trusted user account in Tivoli Access Manager”
2. “Configuring WebSEAL for use with WebSphere Application Server” or “Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server” on page 939
3. “Configuring single signon using the trust association interceptor” on page 940 or “Configuring single signon using trust association interceptor ++” on page 941

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 is the ID and password that WebSEAL uses to identify itself to WebSphere Application Server. To prevent potential vulnerabilities, do not use sec_master as the trusted user account and ensure the password you use is unique and generated randomly. The trusted user account should be used for the TAI or TAI++ only.

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:

```
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” or “Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server” on page 939

Configuring WebSEAL for use with WebSphere Application Server:

A junction must be created between WebSEAL and WebSphere Application Server. This junction will carry the iv-creds (for TAI++) or iv-user (for TAI) and the HTTP basic authentication headers with the request. While WebSEAL can be configured to pass the end user identity in other ways, the iv-creds header is the only one supported by the TAI++ and iv-user the only one supported by TAI.

We recommend that communications over the junction use 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 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 the WebSphere Application Server using the **-c iv-creds** option for TAI++ and **-c iv-user** for TAI. For example (commands are entered as one line):

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 re-create the junction.
2. The junction can be created as `-t tcp` or `-t ssl` depending on your requirements.

For single signon to WebSphere Application Server the SSO 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` and 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 938.
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 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 signon (SSO) user password.

In the following example Tivoli Access Manager plug-in for Web Servers Version 5.1 configuration shows IV headers configured for post-authorization processing and basic authentication configured as the authentication mechanism and for post-authorization processing. After a request has been authorized the basic authentication header is removed from the request (`strip-hdr = always`) and a new one added (`add-hdr = supply`). Included in this new header is the password set when the SSO user was created in "Creating a trusted user account in Tivoli Access Manager" on page 938. This password needs to be specified 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.

An iv-creds header is also added (`generate = iv-creds`) which contains the credential information of the user passed onto WebSphere Application Server. Note also that 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 signon using the trust association interceptor” or “Configuring single signon using trust association interceptor ++” on page 941

Configuring single signon using the trust association interceptor:

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.

This task is performed to enable single signon using the trust association interceptor. The steps involve setting up trust association and creating the interceptor properties.

1. From the WebSphere Application Server console, click **Security > Global security**.
2. Under Authentication mechanisms, click **LTPA**.
3. Under Additional properties, click **Trust association**.
4. Select the **Enable trust association** option.
5. Under Additional properties, click the **Interceptors** link.
6. Click the **com.ibm.ws.security.web.WebSealTrustAssociationInterceptor** link to use the WebSEAL interceptor. This interceptor is the default.
7. Under Additional properties, click **Custom Properties**.
8. Click **New** to enter the property name and value pairs. Ensure the following parameters are set:

Table 14.

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 938 The format of the username is the short name representation. This is a mandatory property. If it is not set in the WebSphere Application Server then TAI initialization will fail.
`com.ibm.websphere.security.webseal.id`	The *iv-user* header, which is com.ibm.websphere.security.webseal.id=iv-user
`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) that are trusted and expected in the request header. For example: com.ibm.websphere.security.webseal.hostnames=host1 This should also include the proxy host names (if any) unless the com.ibm.websphere.security.webseal.ignoreProxy is set to *true*. A list of servers can be obtained using the server list **pdadmin** command.

Table 14. (continued)

Option	Description
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 in the request header. This should also include the proxy ports (if any) 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*.

9. Click **OK**.
10. Save configuration and logout.
11. Restart WebSphere Application Server.

Configuring single signon using trust association interceptor ++:

The following steps are required when setting up security for the first time. Ensure that 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). Save your changes.

This task is performed to enable single signon using trust association interceptor ++. The steps involve setting up trust association and creating the interceptor properties.

1. From the WebSphere Application Server console, click **Security > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**
3. Under Additional properties, click **Trust association**.
4. Select the **Enable Trust Association** option.
5. Click the **Interceptors** link.
6. Click **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** to use the WebSEAL interceptor. This interceptor is the default.
7. Click the **Custom Properties** link.
8. Click **New** to enter the property name and value pairs. Ensure the following parameters are set:

Table 15.

Option	Description
com.ibm.websphere.security.webseal.checkViaHeader	The TAI can be configured 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* the trusted **hostnames** and host **ports** properties do not need to be set. Therefore the only mandatory property when check 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 this property should be set to *false*. **Note:** The via header is part of the standard HTTP header that records the server names the request has 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 938 The format of the username is the short name representation. This is a mandatory property. If it is not set in WebSphere Application Server, then the TAI initialization fails.

Table 15. (continued)

Option	Description
com.ibm.websphere.security.webseal.id	A comma-separated list of headers that should exist in the request. If not all of the configured headers exist in the request then trust can not 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 then TAI initialization will fail.
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 arriving from unlisted ports might not be trusted. If the checkViaHeader property is not set or is set to false then 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 then the TAI initialization fails.

com.ibm.websphere.security.webseal.viaDepth	A positive integer specifying 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 are not trusted then trust cannot be established. The via depth property is used when not all hosts in the via header are required 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 hosts in the via header are checked for trust.

com.ibm.websphere.security.webseal.ssoPwdExpiry	After trust is established for a request the single signon user password is cached saving the need to have the TAI re-authenticate the single signon user with Tivoli Access Manager for every request. The cache timeout period can be modified by setting the single signon password expiry property to the required time in seconds. If the password expiry property is set to 0, the cached password will never expire. 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. It must be remembered 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.

Table 15. (continued)

Option	Description
com.ibm.websphere.security.webseal.configURL	For the TAI to be able to establish trust for a request it requires that SvrSslCfg has been run for the WebSphere Java Virtual Machine resulting in a properties file being created. If this properties file is not at the default URL file://java.home/PdPerm.properties then the correct URL of the properties file must be set in the config 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

9. Click **OK**.
10. Save configuration and logout.
11. Restart WebSphere Application Server.

Global signon principal mapping:

The Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider can be used to manage authentication to WebSphere Enterprise Information Systems (EIS) such as databases, transaction processing systems and message queue systems, located within the WebSphere Application Server security domain. Such authentication is achieved using the Global single signon (GSO) Principal Mapper JAAS login module for J2EE Connector Architecture (J2C) resources.

With GSO principal mapping, a special-purpose JAAS login module inserts a credential into the subject header. This is used by the resource adapter to authenticate to the Enterprise Information System (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 Tivoli Access Manager JACC provider bypasses the credential stored in the XML configuration files and instead uses the Tivoli Access Manager GSO database to provide the EIS security domain authentication information.

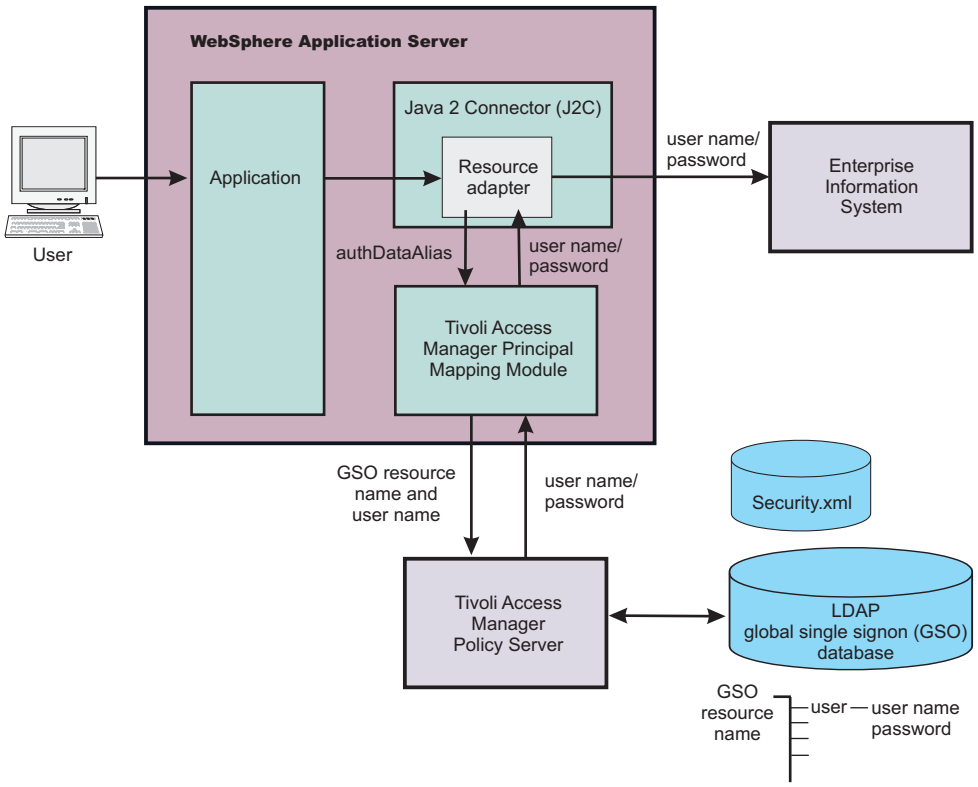
WebSphere Application Server provides a default principal mapping module that associates user credential information with EIS resources. The default mapping module is defined in the WebSphere Application Server administration console on the application login panel. To access the panel, click **Security** > **Global security**. Under JAAS configuration, 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, it contains an alias that refers to a user name and password pair defined elsewhere.

The Tivoli Access Manager Principal Mapping module uses the authDataAlias to determine the GSO resource name and user name required to perform the lookup on the Tivoli Access Manager GSO database. It is the Tivoli Access Manager Policy Server which retrieves the GSO data from the registry.

Tivoli Access Manager stores authentication information on the Tivoli Access Manager GSO database against a resource/user name pair.

GSO principal mapping architecture



Configuring global signon principal mapping:

To create a new application login that uses the Tivoli Access Manager GSO database to store the login credentials:

1. Select **Security** > **Global security**.
2. Under Authentication, click **JAAS Configuration > Application logins**
3. Click **New** to create a new JAAS login configuration.
4. Enter the alias name of the new application login. Click **Apply**.
5. Under Additional properties, click **JAAS Login Modules** link to define the JAAS Login Modules.
6. Click **New** and enter the following:

   Module class name: com.tivoli.pd.as.gso.AMPrincipalMapper
   Use Login Module Proxy: enable
   Authentication strategy: REQUIRED

   Click **Apply**
7. In the *Additional Properties* section, click **Custom Properties** to define Login Module-specific values which are passed directly to the underlying Login Modules.
8. Click **New**.

   The Tivoli Access Manager principal mapping module uses the configuration string, authDataAlias, to retrieve the correct user name and password from the security configuration.

   The authDataAlias passed to the module is configured for the J2C ConnectionFactory. Since the authDataAlias is an arbitrary string entered at configuration time, the following scenarios are possible:

   - The authDataAlias contains both the GSO Resource name and the user name. The format of this string is ″Resource/User″
   - The authDataAlias contains only the GSO Resource name. The user name is determined using the Subject of the current session.

Which scenario to use is determined by a JAAS configuration option. The details of these options are:

Name: com.tivoli.pd.as.gso.AliasContainsUserName

Value: True if the alias contains the user name, false if the user name should be retrieved from the security context.

When entering authDataAliases through the WebSphere Application Server console, the node name is automatically pre-pended to the alias. The JAAS configuration entry is to determine whether this node name should be removed or included as part of the resource name.

Name: com.tivoli.pd.as.gso.AliasContainsNodeName

Value: True if the alias contains the node name.

Enter each new parameter using the following scenario information as a guide.

Note: If the PdPerm.properties configuration file is not located in the default location, JAVA_HOME/PdPerm.properties, then you will also need to add the following property:

Name = com.tivoli.pd.as.gso.AMCfgURL
Value = file:///*path to PdPerm.properties*

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 WAS 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 was 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 WAS 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

You now need to create the J2C authentication aliases. The user name and password assigned to these alias entries is irrelevant as Tivoli Access Manager is responsible for providing user names and

passwords. However, the user name and password assigned to the J2C authentication aliases need to exist so they can be selected for the J2C connection factory in the console.

To create the J2C authentication aliases, from the WebSphere Application Server administrative console, click **Security** >**Global security**. Under **JAAS Configuration** > **J2C Authentication Data** and click **New** for each entry. Refer to the table above for scenario inputs.

The connection factories for each resource adapter that needs to use the GSO database must be configured to use the Tivoli Access Manager Principal Mapping module. To do this:

a. From the WebSphere Application Server console, select **Applications** > **Enterprise Applications** > *application_name*.
b. Under Related items, click **Connector Modules**.
c. Click the **.rar** link.
d. Under Additional properties, click **Resource Adapter** .

   Note: The resource adapter does not need to be packaged with the application. It can be standalone. For such a scenario the resource adapter is configured from **Resources** > **Resource Adapters**.
e. Under Additional properties, click the **J2C Connection Factories** link.
f. Click **New** and enter the connection factory properties.

Note: Configuring custom mapping on connection factory is deprecated in WebSphere Application Server Version 6. To configure the GSO credential mapping, it is recommended that you use the Map Resource References to Resources panel on the administrative console. For more information, refer to “J2EE Connector security” on page 1032.

The Tivoli Access Manager com.tivoli.pd.jcfg.PDJrteCfg utility:
Purpose

Configures and reconfigures the Access Manager Java Runtime Environment component. The Access Manager Java Runtime Environment component enables Java applications to manage and use Tivoli Access Manager security.

Syntax

```
java com.tivoli.pd.jcfg.PDJrteCfg -action {config | unconfig} -cfgfiles_path
configuration_file_path -host policy_server_host -was [-java_home 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
```

-java_home *jre_path*
Specifies the fully-qualified path to the JDK to be configured or reconfigured. If `-java_home` is not specified, the current (default) JDK is used.

For example: `-java_home /usr/lpp/java/J1.3`

-was
Specifies to configure in a WebSphere Application Server environment (as opposed to a Tivoli Access Manager environment).

The following examples demonstrate correct syntax. *Node1* is the name by which the node that contains the administrative server is administered.

Import operation

```
XMLConfig -adminNodeName Node1 -import import.xml
```

Full export operation

```
XMLConfig -adminNodeName Node1 -export export.xml
```

Partial export operation

```
XMLConfig -adminNodeName Node1 -export export.xml -partial imput.xml
```

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.

Using this command does not overwrite JAR files that already exist in the `jre_home\lib\ext` directory, except the `PD.jar` file, which is overwritten if the file exists.

You can install more than one JRE on a given machine. The `pdjrtecfg` command can be used to configure the Access Manager Java Runtime Environment component independently to each of the JREs.

```
${JAVA_HOME}/bin/java
-Dfile.encoding=ISO8859-1 \
-Dws.output.encoding=CP1047 \
-Xnoargsconversion \
-Dpd.home=${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 \
-was
```

The Tivoli Access Manager com.tivoli.pd.jcfg.SvrSslCfg utility:
Purpose

Configures and reconfigures configuration information associated with a Tivoli Access Manager Java application server.

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}
Configures or reconfigures an application server. Options are as follows:

-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.

-action unconfig
Reconfigures an application server to remove the user and server information from the user registry, delete the local key store file, and remove information for this application from the configuration file (without deleting the configuration file). The reconfiguration operation fails only if the caller is unauthorized or the policy server cannot be contacted.

This action can succeed when there is no configuration file. 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.

-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 authorization server.

-cfg_action {create | replace}
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 domain name for the domain to which this server is configured. This domain must exist and an the administrator ID and password must be valid for 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. Do not use relative directory names.

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.

-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
```

User registries

Information about users and groups reside in a user registry.

With WebSphere Application Server, a user registry 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

The users and groups and security role mapping information is used by the configured authorization engine to perform access control decisions.

WebSphere Application Server provides several implementations to support multiple types of operating system base user registries. You can use the custom Lightweight Directory Access Protocol (LDAP) feature to support any LDAP server by setting up the correct configuration (user and group filters). However, support is not extended to these custom LDAP servers because many configuration possibilities exist.

If you are configuring an LDAP registry as the active registry, you can configure one of the following authorization mechanisms:

- System Authorization Facility (SAF) authorization using EJBROLE or GEJBROLE profiles. SAF overrides any other authorization mechanism.
- Tivoli Access Manager as a Java Contract for Containers (JACC) provider. For more information, see “Tivoli Access Manager integration as the JACC provider” on page 1127.

- User-to-role bindings, which are created by the application assembler or the WebSphere Application Server security administrator.

SAF authorization (the use of SAF EJBROLE profiles to assign SAF users and groups to roles) can be used as an authorization mechanism for all user registries. If SAF authorization is selected on the administrative console:

- It overrides any other authorization choice (such as Tivoli Access Manager or SAF authorization).
- You must configure and install a Java Authentication and Authorization Service (JAAS) login mapping module that maps LDAP or custom registry identity to a SAF user ID. For more information, see "Installing and configuring a custom System Authorization Facility mapping module for WebSphere Application Server" on page 1022.

You must provide a mapping from a user registry identity to a SAF user ID unless Local OS is selected as the user registry. For more information, see "Writing a custom System Authorization Facility mapping module for WebSphere Application Server" on page 1012.

Note: These authorization mechanism choices are valid for all user registries, with the exception of Tivoli Access Manager, which is supported for LDAP only.

In addition to Local operating system (OS) and LDAP registries, WebSphere Application Server also provides a plug-in that supports any user registry by using the custom registry feature (also referred to as a *custom user registry*). The custom registry feature supports any user registry that is not implemented by WebSphere Application Server. You can use any registry used in the product environment by implementing the *UserRegistry interface* interface.

The UserRegistry interface is very helpful in situations where the current user and group information exists in some other format (for example, a database) and cannot move to Local OS or LDAP. In such a case, implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all of the security-related operations. Building a custom registry is a software implementation effort; it is expected that the implementation does not depend on other WebSphere Application Server resources, for example, data sources, for its operation.

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.

Steps for selecting a user registry

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.

Before you begin: Before configuring the user registry you need to know the user name (ID) and password to be used, and you must decide which registry to use (Custom, LDAP, or local OS such as SAF-based).

What you need to know: You need to start the Administrative Console by specifying:
`http://server_hostname:9060/ibm/console`

Though different types of registries are supported, only a single active user registry can be configured at once. All the processes in WebSphere Application Server can use one active registry. Configuring the correct registry is a prerequisite to assigning users and groups to roles for applications.

By default, when no registry is configured the Local OS SAF-based registry is used (PQ81586). So if your choice of registry is not Local OS you need to first configure the registry, which is normally done as part of enabling global security, restart the servers, and then assign users and groups to roles for all your applications.

Steps for selecting SAF Authorization:

Before configuring SAF authorization you need to know the user name (ID) and password that are used here. This user can be any valid user in the registry. This user is referred to as either a product security server ID, a server ID or a server user ID in the documentation. Having a server ID means that a user has special privileges when calling protected internal methods.

You need to start the Administrative Console by specifying URL:
`http://server_hostname:9060/ibm/console`

1. Click **Security > Global security**.
2. Under User registries, click **Local OS**.
3. Under General properties, enter the server user ID and server user password. 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 OS user registry to authenticate and obtain privilege information about users by calling the native application programming interfaces (APIs) in that particular user registry.
4. Click **OK**.

Steps for selecting an LDAP user registry:

To use Lightweight Directory Access Protocol (LDAP) as the user registry, you need to know a valid user name (ID), the user password, 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. 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 is used to log into the administrative console once security is turned on. You can use other users to log in if those users are part of the administrative roles.

Perform the following steps to select LDAP as the user registry.

You need to start the administrative console by specifying URL:
`http://server_hostname:9060/ibm/console`

1. Click **Security > Global security** . Under User registries, click **LDAP**.
2. On the LDAP user registry panel in the General Properties section of the Configuration tab, enter the Server user ID and password. 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 OS registry to authenticate and obtain privilege information about users by calling the native application programming interfaces (API) in that particular registry.
3. In the type menu, select the type of LDAP server to which you connect. The type is used to preload default LDAP properties. 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 “Supported directory services” on page 971.
4. In the Host box, enter the host ID (IP address or domain name system (DNS) name) of the LDAP server.
5. In the Port box, enter host port of the LDAP server. The default value is *389*. If multiple WebSphere Application Servers are installed and configured to run in the same single signon 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.

6. In the Base Distinguished Name field, enter the base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service. For example, for a user with a distinguished name (DN) of `cn=John Doe, ou=Rochester, o=IBM, c=US`, you can specify the base DN as (assuming a suffix of c=us): `ou=Rochester,o=IBM,c=us` or `o=IBM,c=us,c=us`. For authorization purposes, this field is case sensitive. This implies that if a token is received (for example, from another cell or 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 field.

   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 run time.

   This field is required for all LDAP directories except for the Domino Directory, where it is optional.
7. In the Bind Distinguished Name field, enter the distinguished name 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 field description for examples of distinguished names.
8. In the Bind Password field, enter the password for the application server to use when binding to the directory service.
9. In the Search Timeout field, enter the timeout value in seconds for an LDAP server to respond before aborting a request. The default value is *300*.
10. Ensure that the **Reuse Connection** option is checked. Enabled (or checked) is the default and specifies that the server should reuse the LDAP connection. Clear this option only in rare situations where a router is used to spray requests to multiple LDAP servers and when the router does not support affinity.
11. The **Ignore Case** option allows you to enable or disable case insensitive authorization check. This field is required when IBM Directory Server is selected as the LDAP directory server. Otherwise, this field is optional and can be enabled when a case sensitive authorization check is required. For example, when you use certificates and the certificate contents do not match the case of the entry in the LDAP server. You c an also enable the Ignore Case field when using single signon (SSO) between the product and Domino. The default is *Disabled*.
12. The **SSL Enabled** option allows you to enable or disable secure socket communication to the LDAP server. When enabled, the LDAP Secure Sockets Layer (SSL) settings are used, if specified.
13. In the SSL Configuration menu, select the Secure Sockets Layer configuration to use for the LDAP connection. This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*.
14. Click **OK**.

Steps for selecting a custom user registry:

Before you begin this task, implement and build the `UserRegistry` interface. For more information on developing custom user registries refer to “Selecting a user registry” on page 848.

Perform the following steps to select a custom user registry.

1. Click **Security > Global security**.
2. Under User registries, click **Custom**.
3. Enter the Server user ID and password in the Server user ID and Server user password fields. 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 OS registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry.

4. Enter a dot-separated class name that implements the `com.ibm.websphere.security.UserRegistry` interface in the Custom registry class name field. Although the custom registry implements the `com.ibm.websphere.security.UserRegistry` interface for backward compatibility, a user registry can alternately implement the `com.ibm.websphere.security.CustomRegistry` interface. The default is `com.ibm.websphere.security.FileRegistrySample`.
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 *%install_root%*`/classes` directory.
6. **Optional:** Select the **Ignore case for authorization** option, which enables WebSphere Application Server to perform a case insensitive authorization check. The default value is *enabled*.
7. Use the Custom Properties link to add any additional properties required to initialize the custom registry. Set the `WAS_UseDisplayName` property, which is predefined by WebSphere Application Server, only when it is required. When the property is set to true, the methods getCallerPrincipal(), getUserPrincipal(), getRemoteUser() methods return the display name. By default, the securityName of the user is returned. This property is primarily introduced to support backward compatibility with the Version 5 custom user registry.
8. Click **OK**.

Configuring user registries

Before configuring the user registry, decide which registry to use. Though different types of registries are supported, all of the processes in WebSphere Application Server can use one active registry. Configuring the correct registry is a prerequisite to assigning users and groups to roles for applications. When a user registry is not configured, the Local OS user registry is used by default. So, if your choice of registry is not Local OS you need to first configure the registry, which is normally done as part of enabling security, restart the servers, and then assign users and groups to roles for all your applications.

After the applications are assigned users and groups, and you need to change the registries (for example from Lightweight Directory Access Protocol (LDAP) to Custom), 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.

Note: If you are switching registries and want to go directly from one user registry to another:

1. Disable security
2. Enable security using the new registry

The following wsadmin command, which uses Jacl, removes all of the users and groups (including the RunAs role) 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 registries.
- The application bindings file does not contain the `accessIDs`, which are unique for each registry even for the same user or group name.

By default, an application does not contain `accessIDs` 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 accessIDs for the applications to improve performance you have to remove the existing user and group information and add the information after configuring the new registry.

For more information on updating accessIDs, see updateAccessIDs in the AdminApp object for scripted administration article.

Complete one of the following steps to configure your user registry:

- "Configuring local operating system user registries" on page 958
- "Configuring Lightweight Directory Access Protocol user registries" on page 961
- "Configuring custom user registries" on page 975.

This step is required as part of enabling security in WebSphere Application Server.

1. If you are enabling security, make sure that you complete the remaining steps. Verify that the Active User Registry field in the **Global Security** panel is set to the appropriate registry. As the final step, validate the user ID and the password by clicking **OK** or **Apply** in the Global Security 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 **OK** or **Apply** in the Global Security panel. After validation, save the configuration, stop and start all WebSphere Application Servers (cells, nodes and all the application servers). To avoid inconsistencies between the WebSphere Application Server processes, make sure that any changes to the registry 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.
3. If System Authorization Facility (SAF) is selected in the new registry, the values in the bindings file are ignored (with the exception of the user ID and password for RunAs role users). Refer to "Updating System Login Configurations to perform a System Authorization Facility identity user mapping" on page 1009 for more information.

Local operating system user registries:

With the local operating system, or Local OS, user registry implementation, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

WebSphere Application Server for z/OS uses the System Authorization Facility (SAF) interfaces. SAF interfaces are defined by MVS to enable applications to use system authorization services or user registries to control access to resources such as data sets and MVS commands. SAF either processes security authorization requests directly or works with RACF, or other security products, to process the requests.

A Local OS user registry is a centralized registry within a sysplex.

Web client certificate authentication is supported when using the local operating system user registry. Digital certificates can be mapped to MVS identities by both Web and Java clients when you select Local OS. A certificate name filter can be used to simplify the mapping. If you are using RACF as the security server, the **RACDCERT MAP** command creates a resource profile that maps multiple user identities to a digital certificate to simplify administration of certificates, conserve storage space in the RACF database, maintain accountability, or maintain access control granularity.

Using both the domain registry and the local registry

When the machine that hosts the WebSphere Application Server process is a member of a domain, both the local and the domain registries are used by default. The following section describes more on this topic and recommends some best practices to avoid unfavorable consequences.

- **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 registry or the domain registry to map to the roles.

In cases where the same users or groups exist in both the local registry and the domain 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 registry tries to authenticate the user first. If the authentication fails the local 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 registry. In case of failure, the local 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, then only that 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.

  Note: A user **Bob** on one machine (the local registry, for example) is not the same as the user **Bob** on another machine (say the domain registry) because the `uniqueID` of **Bob** (the security identifier [SID], in this case) is different in different registries.

- **Examples**

  The machine `MyMachine` is part of the domain `MyDomain`. `MyMachine` contains the following users and groups:
  - MyMachine\user2
  - MyMachine\user3
  - MyMachine\group2

  MyDomain 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 registry is used for authentication. If the authentication fails (the password is different) the local registry is used.
  2. If the user `MyMachine\user2` is mapped to a role, only the `user2` in `MyMachine` has access. So if the `user2` password is the same on both the local and the domain registries, `user2` cannot access the resource, because `user2` is always authenticated using the domain registry. Hence, if both registries have common users, it is recommended that the password be different.
  3. If the `group2` is mapped to a role, only the users who are members of the `MyDomain\group2` can access the resource because `group2` information is first obtained from the domain registry.
  4. If the group `MyMachine\group2` is mapped to a role, only the users who are members of the `MyMachine\group2` can access the resource. A specific group is mapped to the role (`MyMachine\group2` instead of just `group2`).
  5. Use either `user3` or `MyMachine\user3` to map to a role, because `user3` is unique; it exists in one 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.

Note: In a Network Deployment environment, only a centralized repository can be used if more than one node is involved. This usage implies that only the domain registry can be used because the user and group uniqueIDs (SIDs) differ on various nodes, as previously mentioned.

Using either the local or the domain registry. If you want to access users and groups from either the local registry or the domain 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 registry is used. When this property is set to domain, (case insensitive) only the domain registry is used. Set this property by clicking **Custom Properties** in the **Security** > **User Registries** > **Local OS** panel in the administrative console or by using scripts. 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.

Remote registries

By default, the registry is local to all of the product processes. The performance is higher, (no need for remote calls) and the registry also increases availability. Any process failing does not effect other processes.

When using LocalOS as the registry, every product process must run with privilege access.

If this process is not practical in some situations, you can use a remote registry from the node (or in very rare situations from the cell). Using a remote registry affects performance and creates a single point of failure. **Use remote registries only in rare situations.**

The node and the cell processes are meant for manipulating configuration information and for hosting the registry for all the application servers that create traffic and cause problems.

Using a node agent (instead of the cell) to host the remote registry is preferable because the cell process is not designed to be highly available. Also, using a node to host the remote registry indicates that only the application servers in that node are using it. Because the node agent does not contain any application code, giving it the access required privilege is not a concern.

You can set up a remote registry by setting the WAS_UseRemoteRegistry property in the Global Security panel using the **Custom Properties** link at the bottom of the administrative console panel. Use either the`Cell` or the `Node`(case insensitive) value. If the value is `Cell`, the cell registry is used by all of the product processes including the node agent and all of the application servers. If the cell process is down for any reason, restart all of the processes after the cell is restarted. If the node agent registry is used for the remote registry, set the WAS_UseRemoteRegistry value to `node`. In this case, all the application server processes use the node agent registry. In this case, if the node agent fails and does not start automatically, you might need to restart all the application servers after the node agent is started.

Configuring local operating system user registries:

When a Local OS Registry is chosen for z/OS, the started task identity is chosen as the server identity. Thus, a user ID and password is not required to configure the server.

Important: Each started task, (for example, controller, servant, or node agent) might have a different identity. However, note that if you are using the z/OS Customization Dialog the node agent uses the Controller's identity as the for the server identity.

For all servers in a given cell to have the authority needed by the administrative subsystem, they must be part of a common configuration group. This customization is generally provided by the configuration dialogs when WebSphere Application Server for z/OS is initially customized.

The following steps are needed to perform this task initially when setting up security for the first time.

Click **Security > Global security**. Under User registries, click **Local OS**. Under Additional properties, click **z/OS SAF properties**. Select the **Authorization** option.

com.ibm.security.SAF.unauthenticated
This property indicates the MVS user ID that is used to represent unprotected servlet requests and is used for the following functions:

- Authorization if an unprotected servlet invokes an entity bean.
- Identification of an unprotected servlet for invoking a z/OS connector (Customer Information Control System (CICS), Information Management System (IMS)) that uses a current identity when `res-auth=container`.

com.ibm.security.SAF.authorization
This property can be set to `true` or `false`. When this property is set to `true`, SAF EJBROLE profiles are used for user to role authorization for both J2EE applications and the Role-based authorization requests (naming and administration) associated with the WebSphere Application Server run time.

com.ibm.security.SAF.delegation
This property specifies that SAF EJBROLE definitions are to assign which MVS user ID becomes the active identity when you select the RunAs specified role.

com.ibm.security.SAF.EJBROLE.Audit.Messages.Suppress
This property is accessible through the administrative console by completing the following steps:

1. Click **Security > Global security**. Under User Registry, click **Local OS**. Under Additional properties, click **Custom properties > com.ibm.security.SAF.EJBROLE.Audit.Messages.Suppress**.

The property allows you to turn ICH408I messages on or off. The default value for this property is *false*, which does not suppress messages. You can set this value to *true* to suppress the ICH408I messages.

SMF records access violations no matter what value is specified for this new property. This property affects access violation message generation for both application-defined roles and for WebSphere runtime-defined roles for the naming and administrative subsystems. EJBROLE profile checks are done for both declarative (deployment descriptors) and programmatic checks:

- Declarative checks are coded as SecurityConstraints in Web applications, and Deployment Descriptors are coded as SecurityConstraints in EJB files. This property is not used to control messages in this case. Instead, there are a set of roles permitted, and if an access violation occurs an ICH408I access violation message indicates a failure for one of the roles. SMF then logs a single access violation (for that role).
- Program logic checks (or access checks) are performed using the programmatic isCallerinRole(x) for EJB or isUserInRole(x) for Web applications. The com.ibm.security.SAF.EJBROLE.Audit.Messages.Suppress property controls the messages generated by this call.

For more information on SAF authorization, refer to“Controlling access to console users when using a Local OS Registry” on page 849Local OS Registry. For more information on administrative roles, refer to Admin roles.

The Local OS user registry has been configured.

1. If you are enabling security, complete the remaining steps. As the final step, ensure that you validate the user and password by clicking **OK** or **Apply** in the Global Security panel. Save, stop, and start all the product servers.
2. For any changes in this panel to be effective, you need to save, stop and start all the product servers (deployment managers, nodes and Application Servers).

3. If the server comes up without any problems the setup is correct.

Local operating system user registry settings:

Use this page to configure local operating system user registry settings.

To view this administrative console page, click **Security** > **Global Security**. Under User registries, click **Local OS**.

Custom properties

Under the Custom properties link, you can add a value for the com.ibm.security.SAF.EJBROLE.Audit.Messages.Suppress property. Set this property to turn ICH408I messages on or off. The default value for this property is false, which does not suppress messages. You can set this value to `true` to suppress the ICH408I messages.

This property affects access violation message generation for both application-defined roles and for WebSphere Application Server Runtime roles for the naming and administrative subsystems. System Management Facility (SMF) records are unaffected by this property. EJBROLE profile checks are done for both declarative (deployment descriptors) and programmatic checks:

- Declarative checks are coded as security constraints in Web applications, and deployment descriptors are coded as security constraints in enterprise beans. This property is not used to control messages in this case. Instead, a set of roles is permitted, and if an access violation occurs an ICH408I access violation message indicates a failure for one of the roles. SMF then logs a single access violation (for that role).
- Program logic checks (or access checks) are performed using the programmatic isCallerinRole(x) for enterprise bean or isUserInRole(x) for Web applications. The `com.ibm.security.SAF.EJBROLE.Audit.Messages.Suppress` property controls the messages that are generated by this call.

Ignore case for authorization:

When this option is set to true, a case insensitive authorization check is performed.

SAF user IDs are usually in uppercase letters. Enabling this option is necessary only when your registry is case insensitive and does not provide a consistent case when queried for users and groups.

Lightweight Directory Access Protocol:

Lightweight Directory Access Protocol (LDAP) is a user registry in which authentication is performed using an LDAP binding.

WebSphere Application Server security provides and supports 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 (servers) for authenticating a user and other security-related tasks (for example, getting 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 valid user name (ID), the user password, 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. 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, this server ID and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. Choosing an ID and password that do not expire or change often is important. 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 described in Configuring LDAP user registries. 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 Global Security panel). When the server is up, you can change the ID, password and other configuration information and then enable security.

Configuring Lightweight Directory Access Protocol user registries:

Review the article on Lightweight Directory Access Protocol (LDAP) before beginning this task.

1. In the administrative console, click **Security > Global security**.
2. Under User registries, click **LDAP**.
3. Enter a valid user name in the Server user ID 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.
4. Enter the password of the user in the Server user password field.
5. Select the type of LDAP server that is used from the Type list. The type of LDAP server determines the default filters that are used by the 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. If either the IBM Directory Server or the iPlanet Directory Server is selected, also select the Ignore Case field.
6. Enter the fully qualified host name of the LDAP server in the Host field.
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.
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`. This field can be case sensitive. Match the case in your directory server. This field is required for all LDAP directories except the Domino Directory. The Base DN field is optional for the Domino server.
9. Enter the Bind DN name in the Bind distinguished name field, if necessary. 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.
10. Enter the password corresponding to the Bind DN in the Bind password field, if necessary.
11. Modify the Search time out value if required. This timeout value is the maximum amount of time that the LDAP server waits to send a response to the product client before aborting the request. The default is 120 seconds.
12. Deselect the **Reuse connection** option only if you use routers to send requests to multiple LDAP servers, and if the routers do not support affinity. Leave this field enabled for all other situations.
13. Select the **Ignore case for authorization** option, if required. When this flag is enabled, 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 using either the IBM Directory Server or the iPlanet Directory Server LDAP servers, this flag needs enabling because the group information obtained from the LDAP servers is not consistent in case. This inconsistency only effects the authorization check.

14. Enable Secure Sockets Layer (SSL) if the communication to the LDAP server is through SSL. For more information on setting up LDAP for SSL, refer to Configuring SSL for LDAP clients.
15. 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, select the appropriate SSL alias configuration from the list in the SSL configuration field.
16. Click **OK**. The validation of the user, password, and the setup do not take place in this panel. Validation is only done when you click **OK** or **Apply** in the **Global Security** panel. If you are enabling security for the first time, complete the remaining steps and go to the **Global Security** panel. Select **LDAP** as the active user registry. If security is already enabled, but information on this panel changes, go to the **Global Security** panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

Sets the LDAP registry configuration. This step is required to set up the LDAP 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 the final step, validate this setup by clicking **OK** or **Apply** in the Global Security panel.
2. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect.
3. If the server comes up without any problems the setup is correct.

Lightweight Directory Access Protocol 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, click **Security > Global security**. Under User registries, click **LDAP**.

When security is enabled and any of these properties change, go to the Global security panel and click **Apply** to validate the changes.

Server user ID:

Specifies the user ID that is used to run the WebSphere Application Server for security purposes.

Although this ID is not the LDAP administrator user ID, specify a valid entry in the LDAP directory located under the Base Distinguished Name.

Server user password:

Specifies the password corresponding to the security server ID.

Type:

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is supported by WebSphere Application Server for z/OS

For a list of supported LDAP servers, see ″Supported directory services.″ in the documentation.

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 WebSphere Application Servers are installed and configured to run in the same single signon 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 4.0.x configuration, and a WebSphere Application Server at Version 5 is going to interoperate with the Version 4.0.x server, then verify that port `389` is specified explicitly for the Version 5 server.

Default: 389

Base distinguished name (DN):

Specifies the base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service.

For example, for a user with a distinguished name (DN) of `cn=John Doe, ou=Rochester, o=IBM, c=US`, you can specify the base DN as (assuming a suffix of `c=us`): `ou=Rochester, o=IBM, 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 Domino) the base DN in the server must match the base DN from the other cell or Domino server exactly. If case sensitivity is not a consideration for authorization, enable the **Ignore case** field. This field is required for all Lightweight Directory Access Protocol (LDAP) directories except for the Domino Directory, where this field is optional.

If you need to interoperate between WebSphere Application Server Version 5 and a Version 5.0.1 or later server, you must enter a normalized base distinguished name. A normalized base distinguished name does not contain spaces 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`. In WebSphere Application Server, Version 5.0.1 or later, the normalization occurs automatically during run time

Bind distinguished name (DN):

Specifies the distinguished name 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 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 aborting a request.

Default: 120

Reuse connection:

Specifies whether the server reuses the Lightweight Directory Access Protocol (LDAP) connection. Clear this option only in rare situations where a router is used to spray requests to multiple LDAP servers and when the router does not support affinity.

Default: Enabled
Range: Enabled or Disabled

Ignore case for authorization:

Specifies that a case insensitive authorization check is performed when using the default authorization.

This field is required when IBM Tivoli Directory Server is selected as the LDAP directory server.

This field 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.

Otherwise, this field is optional and can be enabled when a case-sensitive authorization check is required. For example, use this field when the certificates and the certificate contents do not match the case used for the entry in the LDAP server. You can enable the **Ignore case** field when using single signon (SSO) between WebSphere 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.

SSL configuration:

Specifies the Secure Sockets Layer configuration to use for the Lightweight Directory Access Protocol (LDAP) connection. This configuration is used only when SSL is enabled for LDAP.

Default: DefaultSSLSettings

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 > Global security**.
2. Under User registries, click **LDAP**.
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 selected in the **LDAP 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 **Global security** 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. It 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. It 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 appear. For example, to display entries of the type `object class = inetOrgPerson` 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 appear. 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 objectclass value is the same objectclass that is defined in the Group Filter, and the property is the member attribute. If the objectclass value does not match the objectclass in 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 (;). They are used to find the group memberships of a user by enumerating all the group attributes 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 objectclass stores the list of members belonging to the group represented by the objectclass. 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. WebSphere security leverages the LDAP server's recursive search functionality to search a user's group memberships, including recursive group memberships. For example:

- IBM Directory Server is pre-configured by WebSphere Application Server security to recursively calculate a user's group memberships using the ibm-allGroup attribute
- SunONE directory server is pre-configured 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 run time, then authentication fails because it results in 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}`

Data type: String

Configuring Lightweight Directory Access Protocol search filters:

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 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.

1. In the administrative console, click **Security > Global security**.
2. Under User registries, click **LDAP**.
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. Also, the filter is used to authenticate a user using 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 LDAP directory service 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 or the groupOfUniqueNames object class, specify the following syntax:

   ```
   (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
   ```

   For more information about this syntax, see the “Supported directory services” on page 971 documentation.
6. Modify the User ID map, if necessary. This filter maps the short name of a user to an LDAP entry. It specifies the piece of information that represents users when these users are displayed with their short names. For example, to display entries of the type 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 user `CN=Bob Smith, ou=austin.ibm.com, o=IBM, c=US` 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 user ID `bob` 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. It 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 belonging 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 objectclass, stores the list of members that belong to the group that is represented by the objectclass. This field takes multiple objectclass:property pairs that are delimited by a semicolon (;). For more information about this syntax, see the "Supported directory services" on page 971.

   For the IBM Tivoli Directory Server, Sun ONE, and Active Directory, this field is used to query all users in a group by using the information that is stored in the user object (instead of querying all the groups individually to find if the user exists in that group). 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 user 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 field on the LDAP settings to make the authorization case insensitive. To access the LDAP setting panel, complete the following steps:

    a. Click **Security > Global security**.

    b. Under User registries, click **LDAP**.
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 LDAP User Registry panel, which contains the previous LDAP directory type, rather than the custom LDAP directory type. Clicking **OK** or **Apply** in the LDAP User 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. To proceed to another panel, use the left navigation. Using the navigation to access the LDAP User Registry panel changes the directory type to `Custom`.
- Choose **Custom** type from the LDAP User 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 (if any) does not take place in this panel. Validation is done when you click **OK** or **Apply** in the Global Security panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Global Security panel. Select **LDAP** as the Active User Registry. If security is already enabled and any information on this panel changes, go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

Setting the LDAP search filters. This step is required to modify existing user and group filters for a particular LDAP directory type. It is also used to set up certificate filters to map certificates to entries in the LDAP server.

1. If you are enabling security, complete the remaining steps. As the final step make sure that you validate this setup by clicking **OK** or **Apply** in the Global Security panel.
2. Save, stop, and start all the product servers (cell, nodes and all 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.

Using specific directory servers as the LDAP server:

For *Using MS Active Directory server as the LDAP server* below, note that 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 possessed by a given user entry, rather than browsing through the member list in each group. If you change this default behavior to browse each group, you can change the **Group Member ID Map** field from `memberof:member` to `group:member`.

Using IBM Tivoli Directory Server as the LDAP server

You can choose the directory type of either **IBM Tivoli Directory Server** or **SecureWay** for the IBM Directory Server.

For supported directory servers, refer to the article, Supported directory services. 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 run time. 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 5.2 on the same machine that you install WebSphere Application Server Version 6.0.x. IBM Tivoli Directory Server Version 5.2 includes WebSphere Application Server Express Version 6.0.x, which the directory server uses for its administrative console. Install the Web Administration tool Version 5.2 and WebSphere Application Server Express Version 6.0.x, which are both bundled with IBM Tivoli Directory Server Version 5.2, on a different machine from WebSphere Application

Server Version 6.0.x. You cannot use WebSphere Application Server Version 6.0.x as the administrative console for IBM Tivoli Directory Server. If IBM Tivoli Directory Server Version 5.2 and WebSphere Application Server Version 6.0.x are installed on the same machine, you might encounter port conflicts.

If you must install IBM Tivoli Directory Server Version 5.2 and WebSphere Application Server Version 6.0.x 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 6.0.x**.
- Install WebSphere Application Server, Version 6.0.x.
- When you install WebSphere Application Server Version 6.0.x, 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 for *WAS_HOME* and *WAS_INSTALL_ROOT*. To change the variables using the administrative console, click **Environment > WebSphere Variables**.

Using a Lotus Domino Enterprise Server as the LDAP server

If you choose the Lotus Domino Enterprise Server Version 6.0.3 or Version 6.5.1 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 has been 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 to the following:

```
User ID Map :    person:shortname
```

Using Sun ONE Directory Server as the LDAP server

You can choose **Sun ONE Directory Server** for your Sun ONE Directory Server system. For supported directory servers, refer to the article, Supported directory services. In Sun ONE Directory Server, the default object class is `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. Thus, create the group from the role. If you want to use `groupOfUniqueName` 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 possessed by a given entry, rather than selecting a group and browsing through the members list. When using roles, you can create a group could be created using a:

- Managed role
- Filtered role
- Nested role

All of these roles are computable by `nsRole` attribute.

Using Microsoft Active Directory server as the LDAP server

To set up Microsoft Active Directory as your LDAP server, complete the following steps.

1. Determine the full distinguished name (DN) and password of an account in the **administrators** group.
2. Determine the short name and password of any account in the Microsoft Active Directory. This password does not have to be the same account that is used in the previous step.

3. Use the WebSphere Application Server administrative console to set up the information needed to use Microsoft Active Directory
   a. Click **Security > Global security**.
   b. Under Authentication, click **Authentication mechanisms > LDAP**.
   c. Set up LDAP with Active Directory at the directory type. Based on the information determined in the previous steps, you can specify the following values on the LDAP settings panel:

      Server user ID
      Specify the short name of the account that was chosen in the second step.

      Server user password
      Specify the password of the account that was chosen in the second step.

      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 was chosen in the first step. For example: `dc=ibm, dc=com`

      Bind distinguished name (DN)
      Specify the full distinguished name of the account that was chosen in the first step. For example: `cn=<adminUsername>, cn=users, dc=ibm, dc=com`

      Bind password
      Specify the password of the account that was chosen in the first step.
   d. Click **OK** to save the changes.
   e. Stop and restart the administrative server so that the changes take effect.
4. **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.
   c. Click **OK** to save the changes
   d. Stop and restart the administrative server so that the changes take effect.

Supported directory services:

WebSphere Application Server security supports several different Lightweight Directory Access Protocol (LDAP) servers. For a list of supported LDAP servers, refer to the **Supported hardware, software and APIs** prerequisite Web site in the “Security: Resources for learning” on page 879 article.

It is expected that other LDAP servers follow the LDAP specification function. 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 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 flag.

The z/OS Security Server LDAP is supported when the DB2 TDBM back end is used. Use the SecureWay Directory Server filters to connect to the z/os LDAP.

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 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 dynamic groups, nested groups, and static 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 Version 6.

Use the direct method for searching group memberships if your LDAP server has such an attribute in 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:

- objectclass:attribute pairs for the indirect method
- attribute:attribute pairs for the direct method

Sample entries of attribute:attribute pairs in Group Member ID Map fields include:

- 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

Sample entries of objectClass:attribute pairs in the Group Member ID Map field include:

- dominoGroup:member for 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, memberOf lists the groups beneath which the group is directly nested only and does not contain the recursive list of nested predecessors. Another example is that the Lotus Domino LDAP server, which 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.

Dynamic groups and nested group support: 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 such so an application does not need to pull 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.

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility. 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 runtime authorization table.

Dynamic and nested group support for the SunONE or iPlanet Directory Server: The SunONE or iPlanet Directory Server uses two grouping mechanisms:

Groups
Groups are entries that name other entries as a list of members or as a filter for members.

Roles Roles are also 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
Entries are members if they match a specified Lightweight Directory Access Protocol (LDAP) filter. In this way, the role depends upon the attributes that are contained in each entry. 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” for more information.

Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server:

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” for more information on this topic.

1. On the Lightweight Directory Access Protocol (LDAP) registry panel, select `SunONE` for the LDAP server.
2. Select the **Ignore case for authorization** option.

3. On the LDAP settings panel, change the Group Filter setting to `&(cn=%v)(objectclass=ldapsubentry))`.
4. On the LDAP settings panel, change the Group Member ID Map setting to `nsRole:nsRole`.

Dynamic groups and nested group support for the IBM Tivoli Directory Server: WebSphere Application Server Version 6 supports all Lightweight Directory Access Protocol (LDAP) dynamic and nested groups when using IBM Tivoli Directory Server 4.1 and later. This function is enabled by default by taking advantage of a new feature in IBM Tivoli Directory Server. IBM Tivoli Directory Server V4.1 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.

Refer to "Configuring dynamic and nested group support for the IBM Tivoli Directory Server" for more information.

Configuring dynamic and nested group support for the IBM Tivoli Directory Server:

When creating groups, ensure that nested and dynamic group memberships work correctly.

1. In the Lightweight Directory Access Protocol (LDAP) user registry configuration panel, select `IBM Tivoli Directory Server` for the LDAP server.
2. On the LDAP settings panel change the Group Filter setting. Change the setting to the following value:

   ```
   (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)
   (objectclass=groupOfURLs))).
   ```
3. On the LDAP settings panel change the Group Member ID Map setting. Change the setting to the following value:

   ```
   ibm-allGroups:member;ibm-allGroups:uniqueMember
   ```
4. On the Add an LDAP entry panel the Auxiliary object class value is `ibm-nestedGroup` when creating a nested group. On the Add an LDAP entry panel, the Auxiliary object class value is `ibm-dynamicGroup` when creating a dynamic group.

Custom user registries:

A *custom user registry* is a customer-implemented user registry, that implements the UserRegistry Java interface, as provided by the product. A custom-implemented user registry can support virtually any type of an account repository from a relational database, flat file, and so on. The custom user registry provides considerable flexibility in adapting product security to various environments where some form of a user registry, other than Lightweight Directory Access Protocol (LDAP) or Local Operating System (LocalOS), already exists in the operational environment.

WebSphere Application Server security provides an implementation that uses various local operating system-based registries (Windows, AIX, Solaris, Linux) and various Lightweight Directory Access Protocol (LDAP)-based registries. However, situations can exist where your user and group data resides in other repositories or custom registries (a database, for example) and moving this information to either a LocalOS or an LDAP registry implementation might not be feasible. For these situations WebSphere Application Server security provides a service provider interface (SPI) that you can implement to interact with your current registry. The SPI is the UserRegistry interface. This interface has a set of methods to implement for the product security to interact with your registries for all security-related tasks. The LocalOS and LDAP registry implementations that are provided also implement this interface. Custom user registries are sometimes called the *pluggable user registries* or *custom registries* for short. Your custom user registry implementation is expected to be thread-safe.

The *UserRegistry interface* is a collection of methods that are required to authenticate individual users using either password or certificates and to collect information about the user (privilege attributes) for

authorization purposes. This interface also includes methods that obtain user and group information so that they can be given access to resources. When implementing the methods in the interface, you must decide how to map the information that is manipulated by the UserRegistry interface to the information in your registry.

Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server components such as data sources, enterprise beans, and so on. Do not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server components during startup. If your previous implementation used these components, make a change that eliminates the dependency.

The methods in the UserRegistry interface operate on the following information for users:

User Security Name
: The user name, which is similar to the user name in the Windows, Linux and UNIX systems Local OS registries. This name is used to log in when prompted by a secured application. By default, the Enterprise JavaBeans (EJB) getCallerPrincipal method and the getRemoteUser and getUserPrincipal servlet methods return this name. The user security name is also referred to as *userSecurityName*, *userName*, or *user name*.

Unique ID
: This ID represents a unique identifier for the user. The UserRegistry interface requires this identifier to be unique. The unique ID similar to the system ID (SID) in Windows systems, the Unique ID (UID) in Linux and UNIX systems, and the distinguished name (DN) in Lightweight Directory Authentication Protocol (LDAP). This ID is also referred to as *uniqueUserId*. The unique ID is used to make the authorization decisions for protected resources.

Display name
: This name is an optional string that describes a user, and it is similar to the FullName attribute in Windows operating systems. The implementation can use display names for informational purposes only; these names are not required to exist or to be unique. The user interface can use the display name to present more information about the user.

Group Security name
: This name, which represents the security group, is also referred to as *groupSecurityName*, *groupName* and *group name*.

Unique ID
: The unique ID is the identifier for a group. This name is also referred to as *uniqueGroupId*.

Display name
: The display name is an optional string that describes a group.

The article on UserRegistry interface describes each of the methods in the UserRegistry interface that need implementing. An explanation of each of the methods and their usage in the sample and any changes from the Version 4 interface are provided. The Related references section provides links to all other custom user registries documentation, including a file-based registry sample. The Sample provided is very simple and is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

Configuring custom user registries:

Before you begin this task, implement and build the UserRegistry interface. For more information on developing custom user registries refer to the article, Developing custom user registries. The following steps are required to configure custom user registries through the administrative console.

1. Click **Security > Global security**
2. Under User registries, click **Custom**.
3. Enter a valid user name in the Server user ID field.
4. Enter the password of the user in the Server user password field.
5. Enter the full name of the location of the implementation class file in the Custom registry class name field as a dot-separated file name. 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 *install_root*/lib/ext directory). This file exists in all the product processes. So, if you are operating in a Network Deployment environment, this file exists in the cell class path and in all of the node class paths.

6. Select the **Ignore case for authorization** option for the authorization to perform a case insensitive check. Enabling this option is necessary only when your 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. Otherwise click **OK** and complete the steps required to turn on security.
8. Enter additional properties to initialize your implementation by clicking **Custom properties**. Click **New**. Enter the property name and value. Click **OK**. Repeat this step to add other additional properties. For the sample, enter the following two properties. It is assumed that the users.props 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 chose and reference their location 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 the “users.props file” on page 1002 and the “groups.props file” on page 1002 article.

To use the users.props and the groups.props files on the z/OS platform, save these files in the ASCII format before calling them from the administrative console.

The Description, Required, and Validation Expression fields are not used and you can leave them blank.

Note: In a Network Deployment environment where multiple WebSphere Application Server processes exist (cell and multiple nodes in different machines), these properties are available for each process. Use the relative name *USER_INSTALL_ROOT* to locate any files, as this name expands to the product installation directory. If this name is not used, ensure that the files exist in the same location in all the nodes. To change the value for the *USER_INSTALL_ROOT* variable

This step is required to set up the custom user registry and to enable security in WebSphere Application Server.

1. Complete the remaining steps, if you are enabling security.
2. After security is turned on, save, stop, and start all the product servers (cell, nodes and all the application servers) for any changes in this panel to take effect.
3. If the server comes up without any problems, the setup is correct.
4. Validate the user and password by clicking **OK** or **Apply** on the Global security panel. Save, synchronize (in the cell environment), stop and restart all the product servers.

UserRegistry.java files:

```
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
// All Rights Reserved * Licensed Materials - Property of IBM
//
// DESCRIPTION:
//
//    This file is the UserRegistry interface that custom registries in WebSphere
//    Application Server implement to enable WebSphere security to use the custom
//    registry.
//
package com.ibm.websphere.security;
```

```
import java.util.*;
import java.rmi.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.cred.WSCredential;/**
 * Implementing this interface enables WebSphere Application Server Security
 * to use custom registries. This interface extends java.rmi.Remote because the
 * registry can be in a remote process.
 *
 * 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 interface UserRegistry extends java.rmi.Remote
{

  /**
   * 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
   * @exception RemoteException
   *    as this extends java.rmi.Remote
   **/
   public void initialize(java.util.Properties props)
      throws CustomRegistryException,
             RemoteException;  /**
   * 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
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String checkPassword(String userSecurityName, String password)
```

```
   throws PasswordCheckFailedException,
          CustomRegistryException,
          RemoteException;  /**
* Maps a certificate (of X509 format) 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
* @exception RemoteException as this extends java.rmi.Remote
**/
public String mapCertificate(X509Certificate[] cert)
   throws CertificateMapNotSupportedException,
          CertificateMapFailedException,
          CustomRegistryException,
          RemoteException;  /**
* 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 OS400 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
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getRealm()
   throws CustomRegistryException,
          RemoteException;  /**
* 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 administrative console and by scripting (command
* line) to make available the users in the registry for adding them (users)
* to roles.
*
* @parameter pattern the pattern to match. (For example., a* will match all
*   userSecurityNames starting with a)
* @parameter 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. 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
* @exception RemoteException as this extends java.rmi.Remote
**/
public Result getUsers(String pattern, int limit)
   throws CustomRegistryException,
          RemoteException;  /**
* Returns the display name for the user specified by userSecurityName.
```

```
*
* This method is called only when the user information displays
* (information purposes only, for example, in the administrative console) and 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 Version 4.0 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 5.0 for the same methods the security
* name is 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 one needs the display name to
* be returned set the property WAS_UseDisplayName to true.
*
* See the documentation for more information.
*
* @parameter 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
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUserDisplayName(String userSecurityName)
   throws EntryNotFoundException,
          CustomRegistryException,
          RemoteException;  /**
* Returns the unique ID for a userSecurityName. This method is called when
* creating a credential for a user.
*
* @parameter userSecurityName the name of the user.
* @return the unique ID of the user. The unique ID for an 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
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUniqueUserId(String userSecurityName)
   throws EntryNotFoundException,
          CustomRegistryException,
          RemoteException;  /**
* Returns the name for a user given its unique ID.
*
* @parameter uniqueUserId the unique ID of the user.
* @return the userSecurityName of the user.
* @exception EntryNotFoundException if the uniqueUserID does not exist.
* @exception CustomRegistryException if there is any registry specific
*            problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUserSecurityName(String uniqueUserId)
   throws EntryNotFoundException,
          CustomRegistryException,
          RemoteException;
```

```
/**
 * Determines if the userSecurityName exists in the registry
 *
 * @parameter userSecurityName the name of the user
 * @return true if the user is valid. false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public boolean isValidUser(String userSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
 * Gets a list of groups that match a pattern in the registy.
 * The maximum number of groups returned is defined by the limit
 * argument.
 * This method is called by the administrative console and scripting
 * (command line) to make available the groups in the registry for adding
 * them (groups) to roles.
 *
 * @parameter pattern the pattern to match. (For e.g., a* will match all
 *   groupSecurityNames starting with a)
 * @parameter limit 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. 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
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public Result getGroups(String pattern, int limit)
    throws CustomRegistryException,
           RemoteException;

/**
 * Returns the display name for the group specified by groupSecurityName.
 *
 * This method may be called only when the group information displayed
 * (for example, the administrative console) and not used in the actual
 * authentication or authorization purposes. If there are no display names
 * in the registry return null or empty string.
 *
 * @parameter 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
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getGroupDisplayName(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the unique ID for a group.
```

```
 * @parameter 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 IDd could
 *  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 EntryNotFoundException,
           CustomRegistryException,
           RemoteException;


/**
 * Returns the unique IDs for all the groups that contain the unique ID of
 * a user.
 * Called during creation of a user's credential.
 *
 * @parameter 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 could be the GID
 *  and the unique ID for the user could be the UID.
 * @exception EntryNotFoundException if unique user ID does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the name for a group given its unique ID.
 *
 * @parameter 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
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Determines if the groupSecurityName exists in the registry
 *
 * @parameter groupSecurityName the name of the group
 * @return true if the groups exists, false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
```

```
 **/
 public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by administrative console and scripting
 * (command line) to verify 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 checks are made on the groups that the role contains.
 *
 * @parameter 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 java.rmi.Remote
 **/
 public List getGroupsForUser(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the limit
 * argument.
 *
 * This method is used by the WebSphere Business Integration
 * Server Foundation process choreographer when staff assignments
 * are modeled using groups.
 *

 * In rare situations if you are working with a registry where getting all of
 * the users from any of your groups is not practical (for example if
 * a large number of users exist) you can throw the NotImplementedException
 * for that particular groups. Make sure that if the WebSphere Business
 * Integration Server Foundation Process Choreographer is installed (or
 * if installed later) that are not modeled using these particular groups.
 * If no concern exists about the staff assignments returning the users from
 * groups in the registry it is recommended that this method be implemented
 * without throwing the NotImplemented exception.
 *
 * @parameter groupSecurityName that represents the name of the group
 * @parameter limit the maximum number of users to return.
 *            This option is very useful in situations where lots of
 *            users are in the registry and getting all of them at
 *            once is not practical. A value of 0 means  get all of
 *            the users and 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 the future.
 * @exception NotImplementedException throw this exception in rare situations
 *            if it is not practical to get this information for any of the
 *            groups from the registry.
 * @exception EntryNotFoundException if the group does not exist in
 *            the registry
```

```
   * @exception CustomRegistryException if any registry-specific
   *            problem occurs
   * @exception RemoteException as this extends java.rmi.Remote interface
   **/
   public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * 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.
   *
   * Note that because this method is not called you can also return the
   * NotImplementedException as the previous documentation says.
   *
   **/
   public com.ibm.websphere.security.cred.WSCredential
                               createCredential(String userSecurityName)
      throws NotImplementedException,
      EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
}
```

FileRegistrySample.java file: The user and group information required by this sample is contained in the "users.props file" on page 1002 and "groups.props file" on page 1002 files.

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, executed, 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 behind this. 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 a 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-
*             specificproblem
**/
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 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, 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 one needs the display name to be returned
* set the property WAS_UseDisplayName to true.
*
*See the InfoCenter 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 an 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 uniqueId.
*
* @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 UniqueId
 * 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 uniqueId.
 *
 * @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 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 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 throw
 * 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 throwing 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 lot 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 throw 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[];

}
```

Result.java file: This module is used by user registries in WebSphere Application Server when calling the getUsers and getGroups methods. The user registries use this method to set the list of users and groups and to indicate if there are more users and groups in the registry than requested.

```
//
//  5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
//  All Rights Reserved * Licensed Materials - Property of IBM
//
package com.ibm.websphere.security;

import java.util.List;

public class Result implements java.io.Serializable {
    /**
      Default constructor
    */
    public Result() {
    }

    /**
       Returns the list of users and groups
       @return the list of users and groups
    */
    public List getList() {
      return list;
    }

    /**
       indicates if there are more users and groups in the registry
    */
    public boolean hasMore() {
      return more;
    }
    /**
       Set the flag to indicate that there are more users and groups
```

```
       in the registry to true
    */
    public void setHasMore() {
      more = true;
    }

    /*
      Set the list of users and groups
      @param list   list of users/groups
    */
    public void setList(List list) {
      this.list = list;
    }

    private boolean more = false;
    private List list;
}
```

Custom user registry settings:

Use this page to configure the custom user registry.

To view this administrative console page, click **Security > Global security**. Under User registries, click **Custom**.

After the properties are set in this panel, click **Apply**. Under Additional Properties, click **Custom properties** to include additional properties that the custom registry requires. The following property is predefined by the product; set this property when required only:

WAS_UseDisplayName
When this property is set to `true`, the getCallerPrincipal(), getUserPrincipal(), and getRemoteUser() methods return the display name. By default, the securityName of the user is returned. This property is introduced to support backward compatibility with the version 4 custom registry.

When security is enabled and any of these custom user registry settings change, go to the Global security panel and click **Apply** to validate the changes.

Server user ID:

Specifies the user ID under which the server runs, for security purposes.

This server ID represents a valid user in the custom registry.

Data type: String

Server user password:

Specifies the password corresponding to the security server ID.

Data type: String

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. Although the custom registry implements the com.ibm.websphere.security.UserRegistry interface, for backward compatibility, a user registry can alternately implement the com.ibm.websphere.security.CustomRegistry interface.

Data type:	String
Default:	com.ibm.websphere.security.FileRegistrySample

Ignore case for authorization:

Specifies that a case insensitive authorization check is performed when you use the default authorization.

Default:	Disabled
Range:	Enabled or Disabled

users.props file: Following is the format for the users.props 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
#
# 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:

```
# 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:
```

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. It 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 and 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 1208 for more information.

The following sections cover the JAAS implementation and programming model:
- “Login configuration for Java Authentication and Authorization Service” on page 1007
- Programmatic login
- “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*. That is, 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 run time automatically determines whether the policy grants the required permission to a specific principal only. If so, the operation is supported if the subject 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 java.security.PrivilegedAction or java.security.PrivilegedExceptionAction. 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 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 specified in the subject when invoking the EJB method within the Subject doAs action block.

This simple rule of letting Subject.doAs overwrite the run-as identity setting is an ideal way to integrate the JAAS programming model with the J2EE run-time environment. However, a general JAAS design oversight was introduced into IBM Software Development Kit (SDK), Java Technology Edition Version 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 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;
   }
});
```

At line three, the subject object is associated with the context of the current thread. As indicated on line 7 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.

Note that 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 specified by the run-as setting and not by the subject identity.

- The subject object 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 representing the caller security identity. The return type differs from the getCallerPrincipal method of the EJBContext interface, which is java.security.Principal.
- The Subject object generated by the J2C DefaultPrincipalMapping module contains a resource principal and a PasswordCredentials list. The resource principal represents the RunAs identity.

Refer to “J2EE Connector security” on page 1032 for more information

Configuring application logins for Java Authentication and Authorization Service

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is a collection of WebSphere Application Server strategic authentication APIs and replaces the Common Object Request Broker Architecture (CORBA) programmatic login APIs.

WebSphere Application Server provides some extensions to JAAS:
- **com.ibm.websphere.security.auth.WSSubject**. The com.ibm.websphere.security.auth.WSSubject API extends the JAAS authorization model to Java 2 Platform, Enterprise Edition (J2EE) resources.
- You can configure JAAS login in the administrative console and store this configuration in the WebSphere configuration application server configuration. However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. If duplicate login configurations are defined in both the WebSphere configuration API and the plain text file format, the one in the WebSphere configuration API takes precedence. Advantages to defining the login configuration in the WebSphere configuration API include:
  – User interface support in defining JAAS login configuration
  – Central management of the JAAS login configuration
  – Distribution of the JAAS login configuration in a Network Deployment product installation

  Due to a design oversight in the JAAS Version 1.0, javax.security.auth.Subject.getSubject() method does not return the subject associated with the running thread inside a java.security.AccessController.doPrivileged() code block. This problem presents an inconsistent behavior that might cause problems. The com.ibm.websphere.security.auth.WSSubject API provides a workaround to associate the subject to a running thread.
- **Proxy LoginModule**. The Proxy LoginModule loads the actual LoginModule. The default JAAS implementation does not use the thread context class loader to load classes. The LoginModule module cannot load if the LoginModule class file is not in the application class loader or the Java extension class loader class path. Due to this class loader visibility problem, WebSphere Application Server provides a proxy LoginModule module to load the JAAS LoginModule using the thread context class loader. You do not need to place the LoginModule implementation on the application class loader or the Java extension class loader class path with this proxy LoginModule module.

  If you do not want to use the Proxy LoginModule, you can place the LoginModule in the `jre/lib/ext` directory. However, this is not recommended due to the security risks.

Two JAAS login configurations are defined in the WebSphere Configuration application programming interface (API) security document for applications to use. Click **Security > Global security**. Under Authentication, click **JAAS configuration > Application logins**. The following JAAS login configurations are available:

ClientContainer

Defines a login configuration and a LoginModule implementation that is similar to that of the WSLogin configuration, but enforces the requirements of the WebSphere Application Server client container. Refer to Configuration entry settings for Java Authentication and Authorization Service for more information.

DefaultPrincipalMapping,
Defines a special LoginModule module that is typically used by J2EE connectors to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back-end enterprise information system (EIS). For more information about J2EE Connector and the DefaultMappingModule module, refer to the J2EE security section.

WSLogin
Defines a login configuration and a LoginModule implementation that applications can use in general.

A new JAAS login configuration can be added and modified using the administrative console. The changes are saved in the cell-level security document and are available to all managed application servers. An application server restart is required for the changes to take effect at run time.

Attention: Do not remove or delete the predefined JAAS login configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them can cause other enterprise applications to fail.

1. Delete a JAAS login configuration.
   a. Click **Security > Global security**.
   b. Under Authentication, click **JAAS Configuration > Application logins**. The Application Login Configuration panel appears.
   c. Select the check box for the login configurations to delete and click **Delete**.
2. Create a new JAAS login configuration.
   a. Click **Security > Global security**.
   b. Click **JAAS Configuration > Application logins**.
   c. Click **New**. The Application Login Configuration panel appears.
   d. Specify the alias name of the new JAAS login configuration and click **Apply**. This value is the name of the login configuration that you pass in the javax.security.auth.login.LoginContext implementation for creating a new LoginContext.

      Click **Apply** to save changes and to add the extra node name that precedes the original alias name. Clicking **OK** does not save the new changes in the `security.xml` file.
   e. Under Additional properties, click **JAAS Login Modules**.
   f. Click **New**.
   g. Specify the Module class name. Specify the WebSphere Proxy LoginModule because of the limitation of the class loader visibility problem.
   h. Specify the LoginModule implementation as the delegate property of the Proxy LoginModule. The WebSphere Proxy LoginModule class name is com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy.
   i. Select **Authentication strategy** from the list and click **Apply**.
   j. Under Additional properties, click **Custom properties**. The Custom properties panel is displayed for the selected LoginModule.
   k. Create a new property with the name `delegate` and the value of the real LoginModule implementation. You can specify other properties like `debug` with the value `true`. These properties are passed to the LoginModule class as options to the initialize() method of the LoginModule instance.
   l. Click **Save**.

   There are several locations within the WebSphere Application Server directory structure where you can place a JAAS login module. The following list provides locations for the JAAS login module in order of recommendation:

   - Within an Enterprise Archive (EAR) file for a specific Java 2 Enterprise Edition (J2EE) application.

     If you place the login module within the EAR file, it is accessible to the specific application only.
   - In the WebSphere Application Server shared library.

If you place the login module in the shared library, you must specify which applications can access the module. For more information on shared libraries, see Managing shared libraries.

- In the Java extensions directory (*WAS_HOME*/jre/lib/ext)

  If you place the JAAS login module in the Java extensions directory, the login module is available to all applications.

  This location is not recommended for WebSphere Application Server for z/OS or WebSphere Business Integration Server Foundation.

Although the Java extensions directory provides the greatest availability for the login module, it is recommended that you place the login module in an application EAR file. If other applications need to access the same login module, consider using shared libraries.

3. Change the plain text file. WebSphere Application Server supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. However, a tool is not provided that edits plain text files in this format. You can define the JAAS login configuration in the plain text file (*install_root*/properties/wsjaas.conf). Any syntax errors can cause the incorrect parsing of the plain JAAS login configuration text file. This problem can cause other applications to fail.

   Java client programs that use the Java Authentication and Authorization Service (JAAS) for authentication must invoke with the JAAS configuration file specified. This configuration file is set in the /*install_root*/bin/launchClient.bat file as `set JAAS_LOGIN_CONFIG=-Djava.security.auth.login.config=%install_root%\properties\wsjaas_client.conf`. If the launchClient.bat file is not used to invoke the Java client program, verify that the appropriate JAAS configuration file is passed to the Java virtual machine with the `-Djava.security.auth.login.config` flag.

A new JAAS login configuration is created or an old JAAS login configuration is removed. An enterprise application can use a newly created JAAS login configuration without restarting the application server process.

However, new JAAS login configurations defined in the *install_root*/properties/wsjaas.conf file, do not refresh automatically. Restart the application servers to validate changes. These JAAS login configurations are specific to a particular node and are not available for other application servers running on other nodes.

Create new JAAS login configurations used by enterprise applications to perform custom authentication. Use these newly defined JAAS login configurations to perform programmatic login.

Login configuration for Java Authentication and Authorization Service:

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. JAAS is WebSphere strategic APIs for authentication and it will replace the CORBA programmatic login APIs. WebSphere Application Server provides some extensions to JAAS:

- **com.ibm.websphere.security.auth.WSSubject**: The com.ibm.websphere.security.auth.WSSubject API extends the JAAS authorization model to J2EE resources. You can configure JAAS login in the administrative console (or by using the scripting functions) and store this configuration in the WebSphere configuration application programming interface (API). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. If duplicate login configurations are defined in both the WebSphere configuration API and the plain text file format, the one in the WebSphere configuration API takes precedence. Advantages to defining the login configuration in the WebSphere configuration API include:
  - User interface support in defining JAAS login configuration
  - Central management of the JAAS login configuration
  - Distribution of the JAAS login configuration in a Network Deployment product installation

  Due to a design oversight in the JAAS 1.0, javax.security.auth.Subject.getSubject() does not return the Subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This can present a inconsistent behavior that might cause problems.

com.ibm.websphere.security.auth.WSSubject provides a work around to associate Subject to thread of execution. com.ibm.websphere.security.auth.WSSubject extends the JAAS authorization model to J2EE resources.

Note: Why WebSphere Application Server has its own subject class: You can retrieve the subjects in a Subject.doAs() block with the Subject.getSubject() call. However, this procedure does not work if there is an AccessController.doPrivileged() call within the Subject.doAs() block. In the following example, s1 is equal to s, but s2 is null:

```
* AccessController.doPrivileged() not only truncates the Subject propagation,
* but also reduces the permissions. It does not include the JAAS security
* policy defined for the principals in the Subject.
Subject.doAs(s, new PrivilegedAction() {
  public Object run() {
    System.out.println("Within Subject.doAsPrivileged()");
    Subject s1 = Subject.getSubject(AccessController.getContext());
    AccessController.doPrivileged(new PrivilegedAction() {
      public Object run() {
      Subject s2 = Subject.getSubject(AccessController.getContext());
      return null;
    }
  });
  return null;
}
});
```

- JAAS Login Configuration can be configured in the administrative console (or by using the scripting functions) and stored in the WebSphere configuration repository. An application can define new JAAS login configuration in the Admin Console and the data is persisted in the configuration respository (stored in the WebSphere configuration API). However, WebSphere still support the default JAAS login configuration format (plan text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WebSphere configuration API and the plan text file format, the one in the WebSphere configuration API takes precedence. There are advantages to defining the login configuration in the WebSphere configuration API:
  - UI support in defining JAAS login configuration.
  - The JAAS configuration login configuration can be managed centrally.
  - The JAAS configuration login configuration is distributed in a Network Deployment installation.
- **Proxy LoginModule**: The Proxy.LoginModule is a proxy to the configured user or the system-defined module that the context class loader uses to load the module instead of the system class loader. The default JAAS implementation does not use the thread context class loader to load classes, the LoginModule could not be loaded if the LoginModule class file is not in the application class loader or the Java extension class loader classpath. Due to this class loader visibility problem, WebSphere provides a proxy LoginModule to load JAAS LoginModule using the thread context class loader. The LoginModule implementation does not have to be placed on the application class loader or the Java extension class loader classpath with this proxy LoginModule.

Note: Do not remove or delete the pre-defined JAAS Login Configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them could cause other enterprise applications to fail.

A system administrator determines the authentication technologies, or LoginModules, to be used for each application and configures them in a login configuration. The source of the configuration information (for example, a file or a database) is up to the current javax.security.auth.login.Configuration implementation. The WebSphere Application Server implementation permits the login configuration to be defined in both the WebSphere configuration API security document and in a JAAS configuration file where the former takes precedence.

JAAS login configurations are defined in the WebSphere configuration API security document for applications to use. To access the configurations, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS configuration > Application logins**.

The **WSLogin** defines a login configuration and LoginModule implementation that may be used by applications in general. The **ClientContainer** defines a login configuration and LoginModule implementation that is similar to that of WSLogin but enforces the requirements of the WebSphere Application Server Client Container. The third entry, **DefaultPrincipalMapping**, defines a special LoginModule that is typically used by Java 2 Connector to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back end enterprise information system (EIS). For more information about Java 2 Connector and the DefaultMappingModule please refer to the Java 2 Security section.

New JAAS login configuration may be added and modified using Security Center. The changes are saved in the cell level security document and are available to all managed application servers. An application server restart is required for the changes to take effect at run time and for the client container login configuration to be made available.

WebSphere Application Server also reads JAAS Configuration information from the wsjaas.conf file under the properties sub directory of the root directory under which WebSphere Application Server is installed. Changes made to the wsjaas.conf file is used only by the local application server and will take effect after restarting the application server. Note that JAAS configuration in the WebSphere configuration API security document takes precedence over that defined in the `wsjaas.conf` file. In other words, a configuration entry in `wsjaas.conf` will be overridden by an entry of the same alias name in the WebSphere configuration API security document.

Note: The Java Authentication and Authorization Service (JAAS) login configuration entries in the Security Center are propagated to the server run time when they are created, not when the configuration is saved. However, the deleted JAAS login configuration entries are not removed from the server run time. To remove the entries, save the new configuration, then stop and restart the server.

The samples gallery provides a JAAS login sample that demonstrates how to use JAAS with WebSphere Application Server. The sample uses a server-side login with JAAS to authenticate a user with the security run time for WebSphere Application Server. The sample demonstrates the following technology:

- Java 2 Platform, Enterprise Edition (J2EE) Java Authentication and Authorization Service (JAAS)
- JAAS for WebSphere Application Server
- WebSphere Application Server security

The form login sample is component of the technology samples. For more information on how to access the form login sample, see Accessing the Samples (Samples Gallery).

Updating System Login Configurations to perform a System Authorization Facility identity user mapping:

To modify configurations to perform System Authorization Facility (SAF) identity mapping (and if WebSphere Application Server is configured), you must take the following:

- Use the WebSphere Application Server administrative console or the scripting tools to update the required system login configurations
- If required, update the appropriate user registry panel to enable SAF authorization

  If you are migrating an existing Network Deployment installation, you must update all the nodes to the service level that supports SAF identity mapping before enabling SAF authorization for a non-local OS registry.

A mapping module must be placed in the Java Authentication and Authorization Service (JAAS) configuration to provide the mapping from a non-local OS registry to a SAF user ID. The `com.ibm.ws.security.common.auth.module.MapPlatformSubject` login module follows this mapping module in the configuration. You can do this using either the Simple WebSphere Authentication Mechanism (SWAM) or the Lightweight Third Party Authentication (LTPA) authentication mechanism. You cannot use an Integrated Cryptographic Service Facility (ICSF) authentication mechanism. Refer to Selecting an authentication mechanism for more information. Refer to Java Authentication and Authorization Service for more information.

Application login configurations do not require changes to modify configurations to perform SAF identity mapping. The WebSphere application login configuration entry WSLogin, calls a system login module that is configured as the default, which performs the mapping if SAF authorization is required.

When LTPA is configured, if you are mapping the WebSphere Application Server user registry to a SAF user ID, the following system login configuration entries must be configured to provide the user mapping:

WEB_INBOUND
: The WEB_INBOUND login configuration handles logins for Web application requests, including servlets and JavaServer pages (JSP). This login configuration interacts with the output object that is generated from a trust association interceptor (TAI) if configured. The Subject that is passed into the WEB_INBOUND login configuration can contain objects that are generated by the TAI.

  WebSphere Application Server administrative console requests and a subset of administrative functions, including file transfer, authenticate using this login configuration entry.

RMI_INBOUND
: The RMI_INBOUND login configuration handles logins for inbound RMI requests. Typically, these logins are requests for authenticated access to Enterprise JavaBeans (EJB) files, and can be performed as Java Management Extensions (JMX) requests when using the RMI connector.

DEFAULT
: The DEFAULT login configuration handles the logins for inbound requests made by most other protocols and internal authentications, such as communication between a z/OS controller and servant processes after an initial authentication request is performed.

When SWAM is configured and you are mapping the WebSphere Application Server user registry to a SAF identity, configure the following system login configuration entry to provide the user mapping:

SWAM_ZOSMAPPING
: This entry is used for all authentication when SWAM is selected.

z/OS System Authorization Facility properties:

Use this page to configure the System Authorization Facility (SAF) properties.

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

1. Click **Global Security > Security**.
2. Under User registries, click **Custom**, **LDAP**, or **Local OS**.
3. Under Additional properties, click **z/OS SAF properties**.

The common properties for unauthenticated user, SAF authorization, and SAF EJBROLE message suppression are no longer custom properties. However, the SAF delegation property continues to be accessible from the local OS user registry.

Unauthenticated user ID: Specify the MVS user ID that is used to represent unprotected servlet requests when SAP authorization is specified or a local OS registry is configured. This property definition is used in the following instances:

- For authorization if an unprotected servlet invokes an entity bean

- For identification of an unprotected servlet for invoking a z/OS connector such as Customer Information Control System (CICS) or Information Management System (IMS) that uses a current identity when `res-auth=container`
- When an application-initiated Synch to OS thread function is attempted (Refer to ″Understanding application Synch to OS Thread Allowed″ and ″When to use application Synch to OS Thread Allowed and When to use application Synch to OS Thread Allowed articles for more information.″)

Authorization: Select this option to specify that SAF EJBROLE profiles are used for user-to-role authorization for both Java 2 Platform, Enterprise Edition (J2EE) applications and the role-based authorization requests (naming and administration) that are associated with WebSphere Application Server run time.

If a Lightweight Access Directory Protocol (LDAP) registry or Custom registry is configured and SAF authorization is specified, a mapping to a z/OS principal is required at each login for any protected methods to run:

- If the authentication mechanism is Lightweight Third Party Authentication (LTPA), it is recommended that you update all of the following configuration entries to include a mapping to a valid z/OS principal (such as WEB_INBOUND, RMI_INBOUND, and DEFAULT).
- If the authentication mechanism is Simple WebSphere Authentication Mechanism (SWAM), you must update the SWAM_ZOSMAPPING configuration entry to include a mapping to a valid z/OS principal.

Using Pluggable Login Modules to perform Java 2 Platform, Enterprise Edition identity to Resource Access Control Facility user mapping:

Configure the active WebSphere Application Server User Registry as an Lightweight Directory Access Protocol (LDAP) registry or a Custom registry, and use Resource Access Control Facility (RACF) and System Authorization Facility (SAF) services such as:

- System Authorization Facility (SAF) EJBROLE profiles to control WebSphere Application Server authorization. Refer to Role-based authorization for more information.
- Enabling an application to run a WebSphere Application Server application and set the operating system (OS) identity to match the J2EE identity. This is known as application Sync to OS Thread. Refer to “Understanding application Synch to OS Thread Allowed” on page 894 and “When to use application Synch to OS Thread Allowed” on page 896 for more information.
- Using the J2EE client identity as the identity when issuing a Connection Management request for a local native connector such as CICS, Information Management System (IMS), Database 2 (DB2), or Java Messaging Service (JMS). Refer to “Understanding Java 2 Platform, Enterprise Edition identity and an operating system thread identity” on page 896 for more information.
- Auditing using SMF audit. Refer to Overview of SMF record type 80 for more information.

You must configure a pluggable mapping module followed by a WebSphere Application Server for z/OS-supplied module in appropriate system login configurations to use pluggable login modules.

If a registry other than local OS is selected and no mapping is done or no valid mapping is available for a particular identity:

- *SAF authorization is not supported*: If SAF authorization is selected and a method is protected the method fails.
- *Application Synch to OS thread is not supported*: Requests always run using the user ID of the servant.
- When `res-auth=container` is specified to native connectors and no alias is identified, a connection management request runs under the servant user ID

Pluggable login modules can be used when:

- The WebSphere Application Server authentication mechanism specified is Simple WebSphere Authentication Mechanism (SWAM) or Lightweight Third-Party Authentication (LTPA)

- The Internet Inter-ORB protocol (IIOP) authentication protocol negotiated uses Common Secure Interoperability Version 2 (CSIV2)
- A Web request is issued

Writing a custom System Authorization Facility mapping module for WebSphere Application Server:

You can customize Java Authentication and Authorization (JAAS) login configurations by writing a customized login mapping module.

The WebSphere Application Server `ltpaLoginModule` module and the `AuthenLoginModule` module use the shared state to save state information with the capability to allow `LoginModules` can modify state information. The `ltpaLoginModule` initializes the callback array in the `login()` method using the following code. The callback array is created by `ltpaLoginModule` only if an array is not defined in the shared state area.

In the following code example, the reliance is made on the availability of a Java 2 Platform, Enterprise Edition (J2EE) identity to control the mapping to a System Authorization Facility (SAF) identity. This code uses the `Constants.WSPRINCIPAL_KEY` value in the shared state. The value is placed in the code by a WebSphere Application Server 1 login module. You can insert a custom `LoginModule` after the `ltpaLoginModule` module, and just before the `MapPlatformSubject` module. If you do this, use a callback array or other shared state values to obtain a value used to control the mapping to the z/OS user ID.

The following is a sample `SAFMappingModule`:

```
//
// This program may be used, executed, copied, modified and
// distributed without royalty for the purpose of developing,
// using, marketing, or distributing.
//
//

package com.ibm.websphere.security;

import com.ibm.websphere.security.auth.CredentialDestroyedException;
import com.ibm.websphere.security.auth.WSPrincipal;
import com.ibm.websphere.security.cred.WSCredential;
import com.ibm.wsspi.security.auth.callback.Constants;
import com.ibm.wsspi.security.token.AttributeNameConstants;
import java.lang.reflect.Array;
import java.util.Map;
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.CredentialExpiredException;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;

/**
 *
 * SampleSAFMappingModule demonstrates a custom login module
 * that maps the existing WSPrincipal from the shared state to a z/OS
 * user id.
 *
 *
 *
 * The following values will be set into the shared state if authentication
 * succeeds.  If authentication fails, this login module will still indicate
 * success, but no values are set into the shared state.
 *
 *   AttributeNameConstants.ZOS_USERID
 *   AttributeNameConstants.ZOS_AUDIT_STRING
 *   AttributeNameConstants.CALLER_PRINCIPAL_CLASS
 *
 * This login module does not use any callbacks, nor does it modify the Subject
```

```
 * in any way.
 *
 * @author IBM Corporation
 * @version 1.0
 * @since 1.0
 */
public class SampleSAFMappingModule implements LoginModule
{
    /*
     * Constant that represents the name of this mapping module.  Whenever this sample
     * code is used to create a class with a different name, this value should be changed.
     *
     * By default, this value is used as part of the sample audit token, and for debugging
     * purposes.
     */
    private final static String MAPPING_MODULE_NAME = "com.ibm.websphere.security.SampleSAFMappingModule";

    /*
     * Constant that represents the maximum length of the ZOS_USERID.  Current MVS naming
     * restrictions limit this to eight characters.
     *
     * When the option to useWSPrincipalName is chosen, the name from the WSPrincipal is
     * shortened to this many characters.
     */
    private final static int MAXIMUM_NAME_LENGTH = 8;

    /*
     * Specifies whether or not to use this module's default mapping behavior, which is
     * to use the WSPrincipal name to generate the ZOS_USERID.  This depends on the
     * value of the "useWSPrincipalName" option passed in from the LoginContext.
      */
    private boolean useWSPrincipalName = true;

    /*
     * Specifies whether debugging is enabled for this Login Module.  This depends on the
     * value of the "debug" option passed in from the LoginContext.
     */
    private boolean debugEnabled = false;

    /*
     * Stores the Subject passed from the LoginContext.
     */
    private Subject subject;

    /*
     * Stores the CallbackHandler passed from the LoginContext.
      */
    private CallbackHandler callbackHandler;

    /*
     * Stores the shared state Map passed from the LoginContext.
     */
    private Map sharedState;

    /*
     * Stores the options Map passed from the LoginContext.
     */
    private Map options;

    /*
     * This value is used to store the success or failure of the login() method so
     * that commit() and abort() can act differently in the two cases if so desired.
     */
    private boolean succeeded = false;

    /**
     * Construct an uninitialized mapping module object.
```

```
 */
public SampleSAFMappingModule()
{
}

/**
 * Initialize this login module.
 *
 * This is called by the LoginContext after this login module is
 * instantiated. The relevant information is passed from the LoginContext
 * to this login module. If the login module does not understand any of the data
 * stored in the sharedState and options parameters,
 * they can be ignored.
 *
 * @param subject
 *                The subject that this LoginContext is authenticating
 * @param callbackHandler
 *                A CallbackHandler for communicating with the end user
 *                to gather login information (e.g., username and password).
 * @param sharedState
 *                The state shared with other configured login modules.
 * @param options
 *                The options specified in the login configuration for this particular login module.
 */
public void initialize(Subject newSubject, CallbackHandler newCallbackHandler,
Map newSharedState, Map newOptions)
{
    // obtain the value for debug before anything else so that tracing can be used within
    // this method
    if (newOptions.containsKey("debug"))
    {
        String debugEnabledString = (String) newOptions.get("debug");
        if (debugEnabledString != null && debugEnabledString.toLowerCase().equals("true"))
        {
            debugEnabled = true;
        }
    }

    if (debugEnabled)
    {
        debug("initialize() entry");
    }

    // this login module is not going to use any of these objects except for the sharedState,
    // but for consistency with most login modules, we will save a reference to all of them
    this.subject = newSubject;
    this.callbackHandler = newCallbackHandler;
    this.sharedState = newSharedState;
    this.options = newOptions;

    if (options.containsKey("useWSPrincipalName"))
    {
        String useWSPrincipalNameString = (String) options.get("useWSPrincipalName");
        if (useWSPrincipalNameString != null && useWSPrincipalNameString.toLowerCase().equals("false"))
        {
            useWSPrincipalName = false;
        }
    }

    if (debugEnabled)
    {
        debug(new Object[] { "initialize() exit", subject, callbackHandler, sharedState, options });
    }
}

/**
 * Method to map the WSPrincipal to a ZOS_USERID
```

```
 *
 * This method derives a ZOS_USERID and stores it into the Shared State for use by a later
 * Login Module.
 *
 * @return true if the authentication succeeded, or false
 *         if this Login Module should be ignored
 * @exception LoginException
 *         if the authentication fails, which is impossible for this Login Module
 */
public boolean login() throws LoginException
{
    if (debugEnabled)
    {
        debug("login() entry");
    }

    if (sharedState.containsKey(AttributeNameConstants.ZOS_USERID))
    {
        // we don't want to override this value if, for whatever reason, another Login Module
        // has already placed it into the shared state, but we still consider this a success
        // because the exit criteria for this module has been met
        if (debugEnabled)
        {
            debug("ZOS_USERID already exists:  so no additional work is needed");
        }
        succeeded = true;
    }
    else if (!sharedState.containsKey(Constants.WSPRINCIPAL_KEY) ||
             !sharedState.containsKey(Constants.WSCREDENTIAL_KEY))
    {
        // if there isn't a Principal or Credential in the shared state, we can't do
        // anything so we'll return false to inform the LoginContext to ignore this module
        if (debugEnabled)
        {
            debug("Principal or Credential is unavailable:  skipping this Login Module");
        }
        succeeded = false;
    }
    else
    {
        if (debugEnabled)
        {
            debug("Principal and Credential are available:  continue with login");
        }

        String name = null;
        String audit = null;
        String principalClass = null;

        // extract the WSPrincipal and WSCredential from the shared state
        WSPrincipal principal = (WSPrincipal) sharedState.get(Constants.WSPRINCIPAL_KEY);
        WSCredential credential = (WSCredential) sharedState.get(Constants.WSCREDENTIAL_KEY);

        if (useWSPrincipalName)
        {
            // this sample mapping module provides a method to obtain the ZOS_USERID directly
            // from the WSPrincipal name if the property "useWSPrincipalName" is set to true
            if (debugEnabled)
            {
                debug("Using name from WSPrincipal to obtain ZOS_USERID");
            }

            name = createName(principal);

            String realm = getRealm(credential);

            // for this example, a sample audit token will be created that combines the ZOS_USERID,
```

```
            // the realm, and the name of this mapping module
            //
            // a custom audit token can be created using any available data rather than using
            // this sample token
            audit = realm + "/" + name + " MappingModule:" + MAPPING_MODULE_NAME;

            // A Subject may contain more than one Principal.  This value specifies the
            // class of the Principal to be returned when the Subject is asked for the
            // Caller Principal.  If a custom Principal class has been placed into the
            // Subject, that class name can be specified here.
            //
            // Two predefined values for the Caller Principal Class are provided:
            //
            // - AttributeNameConstants.ZOS_CALLER_PRINCIPAL_CLASS
            //   the z/OS Principal class
            //
            // - AttributeNameConstants.DEFAULT_CALLER_PRINCIPAL_CLASS
            //   the default Principal class
            principalClass = AttributeNameConstants.DEFAULT_CALLER_PRINCIPAL_CLASS;
            succeeded = true;
        }
        else
        {
            if (debugEnabled)
            {
                debug("Using Custom logic to obtain ZOS_USERID");
            }
            // if the behavior provided by this mapping module to obtain the ZOS_USERID from the
            // WSPrincipal name is not enough, custom mapping logic can be provided here
            //
            // to use this custom mapping logic, the property "useWSPrincipalName" must be set
            // to false

            // name = ...custom logic
            // audit = ...custom logic
            // principalClass = ...custom logic

            // by default, no custom mapping is provided, so the success of this code path
            // is false; if custom mapping is provided, the following variable should be
            // modified to represent the success or failure of the custom mapping
            succeeded = false;
        }

        if (succeeded)
        {
            // now that we have values for name, audit, and principalClass, we just need to set
            // them into the shared state
            sharedState.put(AttributeNameConstants.ZOS_USERID, name);
            sharedState.put(AttributeNameConstants.ZOS_AUDIT_STRING, audit);
            sharedState.put(AttributeNameConstants.CALLER_PRINCIPAL_CLASS, principalClass);
            if (debugEnabled)
            {
                debug(new Object[] { "Values have been stored into the shared state", name,
                      audit, principalClass });
            }
        }
    }

    if (debugEnabled)
    {
        debug("login() exit");
    }
    return succeeded;
}

/**
 * Method to commit the authentication result.
```

```
 *
 * This Login Module does not need to commit any data, so we will simply return.
 *
 * @return true if the original login succeeded, or false
 *         if the original login failed
 * @exception LoginException
 *         if the commit fails, which cannot happen in this Login Module
 */
public boolean commit() throws LoginException
{
    if (debugEnabled)
    {
        debug("commit() entry");
    }

    // the return value of commit() is the same as the success of the original login
    boolean returnVal = succeeded;

    cleanup();

    if (debugEnabled)
    {
        debug("commit() exit");
    }
    return returnVal;
}

/**
 * Method to abort the authentication process (Phase 2).
 *
 * No matter whether our original login succeeded or failed, this method cleans up
 * our state and returns.
 *
 * @return true if the original login succeeded, or false
 *         if the original login failed
 * @exception LoginException
 *         if the abort fails, which cannot happen in this Login Module
 */
public boolean abort() throws LoginException
{
    if (debugEnabled)
    {
        debug("abort() entry");
    }

    // the return value of abort() is the same as the success of the original login
    boolean returnVal = succeeded;

    cleanup();

    if (debugEnabled)
    {
        debug("abort() exit");
    }
    return returnVal;
}

/**
 * Method which logs out a Subject.
 *
 * Since our commit method did not modify the Subject, we don't have anything to
 * logout or clean up and can just return true.
 *
 * @return true if the logout succeeded
 * @exception LoginException
 *         if the logout fails, which cannot happen in the Login Module
 */
```

```
public boolean logout() throws LoginException
{
    if (debugEnabled)
    {
        debug("logout() entry");
    }

    // our local variables were cleanup up during the commit, so no further cleanup is needed

    if (debugEnabled)
    {
        debug("logout() exit");
    }

    // since there is nothing to logout, we always succeed
    return true;
}

/*
 * Cleans up our local variables; the only cleanup required for
 * this Login Module is to set our success variable back to false.
 */
private void cleanup()
{
    if (debugEnabled)
    {
        debug("cleanup() entry");
    }

    // there's nothing to cleanup, really, so just reset our success variable
    succeeded = false;

    if (debugEnabled)
    {
        debug("cleanup() exit");
    }
}

/*
 * Private method to print trace information.  This implementation uses System.out
 * to print trace information to standard output, but a custom tracing system can
 * be implemented here as well.
 */
private void debug(Object o)
{
    System.out.println("Debug: " + MAPPING_MODULE_NAME);
    if (o != null)
    {
        if (o.getClass().isArray())
        {
            int length = Array.getLength(o);
            for (int i = 0; i < length; i++)
            {
                System.out.println("\t" + Array.get(o, i));
            }
        }
        else
        {
            System.out.println("\t" + o);
        }
    }
}

/*
 * Private method to obtain the realm name from the Credential and return it.  This
 * keeps the exception handling involved with obtaining the realm name out of the main
 * login() logic.
```

```
 */
private String getRealm(WSCredential credential)
{
    if (debugEnabled)
    {
        debug("getRealm() entry");
    }

    String realm = null;

    try
    {
        realm = credential.getRealmName();

        if (debugEnabled)
        {
            debug("Got realm='" + realm + "' from credential");
        }
    }
    catch (Exception e)
    {
         // getRealmName throws CredentialExpiredException and CredentialDestroyedException
         if (debugEnabled)
         {
             debug(new Object[] { "Caught exception in getRealm: ", e });
         }
         realm = "UNKNOWN_REALM";
    }

    if (debugEnabled)
    {
        debug("getRealm() exit");
    }
    return realm;
}

/*
 * Private method to generate the ZOS_USERID from the WSPrincipal name.
 */
private String createName(WSPrincipal principal)
{
    if (debugEnabled)
    {
        debug("createName() entry");
    }

    String name = principal.getName();

    if (debugEnabled)
    {
        debug("Using name='" + name + "' from principal");
    }

    // WSPrincipal.getName() might return REALM/NAME, so parse the String to obtain just the name
    int index = name.indexOf("/") + 1; // index of the first character after the first /
    if (index >= name.length())
    {
        // this block handles the case where the first / is the last character in the String,
        // it really shouldn't happen, but if it does we can just strip it off
        name = name.substring(0, index - 1);

        if (debugEnabled)
        {
            debug("Stripping trailing / from WSPrincipal name");
        }
    }
    else
```

```
        {
            // index is either 0 (if no / exists in the name), or it is the position
            // after the first / in the name
            //
            // either way, we will take the substring from that point until the end of the string
            name = name.substring(index);
        }

        // shorten the name if its length exceeds the defined maximum
        if (name.length() > MAXIMUM_NAME_LENGTH)
        {
            name = name.substring(0, MAXIMUM_NAME_LENGTH);

            if (debugEnabled)
            {
                debug("WSPrincipal name shortened to " + name);
            }
        }

        // MVS ids are all upper case
        name = name.toUpperCase();

        if (debugEnabled)
        {
            debug("createName() exit");
        }
        return name;
    }
}
```

The sample mapping module creates a mapping between the Lightweight Directory Access Protocol (LDAP) identity and the z/OS identity. The LDAP identity is used as the z/OS user ID. If a different mapping is required this module can be customized (as shown in the else clause) to perform the mapping. Then:

1. Compile the Java code. Make sure that the code is trusted and treated with the same care as an APF-authorized module. The default Java Authorization and Authentication Service (JAAS) System login configuration is accessed from the z/OS controller.
2. If you specify a mapping class other than the default supplied by IBM, you must install the class into the classes directory of the Application Server and Deployment Managers. Place the Java archive (JAR) file into the *WAS_HOME*/classes directory for each node in the cell, including the deployment manager node in a Network Deployment cell.

If a non-Local OS registry is configured and the Authorization option is selected, you must install a mapping class followed by the `com.ibm.ws.security.common.auth.module.MapPlatformSubject` login module. The mapping class must be placed in the JAAS configuration in order to provide mapping from a registry other than Local OS to a SAF user ID prior to enabling global security. The Authorization option is accessible by completing the following steps:

1. Click **Security > Global security**.
2. Under User registries, click **Custom** or **LDAP**.
3. Under Additional properties, click **z/OS SAF properties**.

For more information JAAS and SAF, see “Configuring application logins for Java Authentication and Authorization Service” on page 1005 and “Updating System Login Configurations to perform a System Authorization Facility identity user mapping” on page 1009.

Using a Java Authentication and Authorization Services login module to map a registry principal to a System Authorization Facility user ID:

A Java Authentication and Authorization Services (JAAS) login module can be used to map a registry principal to the System Authorization Facility (SAF) user ID.

- A pluggable login module can set z/OS well-defined attributes in the shared map during login.
- A sample mapping module (supplied by WebSphere Application Server) `com.ibm.websphere.security.SampleSAFMappingModule` is provided. This module sets the z/OS attributes defined in the Shared State. This module must precede the `com.ibm.ws.security.common.auth.module.MapPlatformSubject` mapping module entry in the list of login modules.

The set of well-defined attributes that are used in WebSphere Application Server mapping are defined in the `com.ibm.wsspi.security.token.AttributeNameConstants` class available in the `sas.jar` file:

com.ibm.wsspi.security.token.AttributeNameConstants.ZOS_USERID
Use this attribute to set the value of the MVS user ID used when an operation is performed that requires a z/OS SAF user ID. If no value is specified, WebSphere Application Server uses the unauthenticated user to establish a SAF user ID. This SAF user ID must be a valid MVS user ID.

com.ibm.wsspi.security.token.AttributeNameConstants.ZOS_AUDIT_STRING
Use this attribute to indicate that the specified string is placed in the `X500Name` property when creating a Resource Access Control Facility (RACF) access control environment element (ACEE).

This associates an audit string with a SAF user, which is displayed in a System Management Facility (SMF) record when either one of the following is performed:

- EJBROLE authorization check
- Any access check for an application running with the operating system identity synchronized to the Java 2, Enterprise Edition (J2EE) identity (Refer to Synchronizing a Java thread identity and an operating system identity for more information.)

A maximum of 223 characters can be placed in this field. If the specified value is larger than 223 characters, only the first 223 characters are used. If this value is omitted, no audit data is added when building a principal. Any audit data recorded in this field is prefixed within the SMF audit record string ″WebSphere Mapped Userid″.

com.ibm.wsspi.security.token.AttributeName.Constants.CALLER_PRINCIPAL_CLASS
Use this optional field to indicate which principal class (in a JAAS subject) is returned when using the getCallerPrincipal and getUserPrincipal APIs. This principal can be created by either of the following:

- WebSphere Application Server runtime
- A JAAS login module

The default value of this field is `com.ibm.websphere.security.auth.WSPrincipal`. Using this default value returns the WebSphere Application Server principal name in the configured WebSphere Application Server registry.

To return a mapped SAF principal, specify `com.ibm.ws.security.zos.Principal`. If a value is specified but no principal matches the specified `CALLER_PRINCIPAL_CLASS` value, the return value indicates an unauthenticated user. Specifying `getUserInRole` returns a null value, and specifying `getCallerPrincipal()` returns a string that indicates that the user is unauthenticated.

Note: Some network identities are not processed using the mapping module provided:

Server identity
This identity is always mapped to the user ID of the process and is assigned by the STARTED profile.

SAF identity corresponding to the UNAUTHENTICATED user
The SAF identity corresponding to the UNAUTHENTICATED user means there is no network identity. This value is configured using the customization dialogs, and can be modified using the administrative console. It is recommended that you create the SAF identity for unauthenticated users with the RESTRICTED attribute.

Refer to Understanding Java 2 Platform, Enterprise Edition identities and operating system thread identities for more information.

Installing and configuring a custom System Authorization Facility mapping module for WebSphere Application Server:

In order to use a pluggable login module to perform Java 2 Platform, Enterprise edition (J2EE) identity to Resource Access Control Facility (RACF) user mapping, a pluggable mapping module followed by a WebSphere Application Server for zOS-supplied module must be configured in appropriate Java Authentication and Authorization Service (JAAS) system login configurations. This enables an installation to configure the active WebSphere Application Server user registry as either Lightweight Directory Access Protocol (LDAP) or a Custom registry and use System Authorization Facility (SAF) authorization.

Before proceeding you should make sure you know how to write a mapping module to get a SAF identity. For more information, refer to Writing a custom System Authorization Facility mapping module for WebSphere Application Server. If you use anything other than the sample, you must build the relevant classes and install it into the `<WAS_HOME>/classes` directory for each node in the cell, including the deployment manager node in a Network Deployment cell. If Java 2 security is enabled, ensure that the server.policy file is updated to provide appropriate permissions.

The custom SAF mapping module (either `com.ibm.websphere.security.SampleSAFMappingModule` or a customer-written mapping module) must be added to each of the system login module entries below, and must be changed to the second to last position in the order manually for the system login modules as indicated below.

- For Simple WebSphere Authentication Mechanism (SWAM), add the entry to the `SWAM_ZOSMAPPING` login module.
- For Lightweight Third Party Authentication (LTPA), add the entry to the `WEB_INBOUND`, `RMI_BOUND`, and `DEFAULT` login modules.

Note: For base configuration, if you select SWAM as your authentication mechanism, update the `SWAM_ZOSMAPPING` entry. However, if you plan to use LTPA as your authentication mechanism, set up all four system login module entries. For an ND configuration you only need to configure the LTPA authentication mechanism configuration entries.

To add a custom SAF mapping module to one of the system login modules listed above, log on to the administrative console application and:

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS configuration > System logins >** *login_module_name*.
3. Under Additional properties, click **JAAS login modules > New**.
4. Enter the class name of the custom login module in the *Module Classname* file. (Use `com.ibm.websphere.security.SampleSAFMappingModule` for the shipped sample module).
5. Click **Apply** to add the new module to the login module list.
6. Click **Security > Global security**.
7. Under Authentication, click **JAAS Configuration > System logins >** *login_module_name*.
8. Under Additional properties, click **JAAS login modules > Set order**. The new mapping module is probably at the end of the list, and must come before `com.ibm.ws.security.common.auth.module.MapPlatformSubject`.
9. Select the box next to the new mapping module and then click **Move up**. When the mapping modules are in the correct order, click **Apply**, then **Save**, and **Save** (be sure to select *Synchronize changes with Nodes* if you are working with a Network Deployment cell).

Make these changes for each of the system login modules needed for your WebSphere Application Server for z/OS configuration. The choice of which system login modules are needed is based on your authentication mechanism (SWAM or LTPA).

Note: If the SAF identity mapping module you installed has configurable properties, you can update them by creating custom properties in the system login panel in the administrative console. Use this example to update properties if you used the `SampleSAFMapping` module as a prototype and updated the **else** clause to provide custom mapping logic. In this case you must create the `useWSPrincipleName` custom property and set it to **false** for each affected JAAS login configuration that uses the modified `SampleSAFMappingModule`.

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS configuration > System logins >***login_module_name*.
3. Under Additional properties, click **JAAS login modules > `com.ibm.websphere.security.SampleSAFMappingModule`**.
4. Under Additional properties, click **Custom Properties > New**.
5. Enter the custom property name `useWSPrincipalName` and the value **false**.
6. Click **Apply**, **Save**, and **Save**.

Repeat this process for each of the system login modules that use the modified `SampleSAFMappingModule`.

For more information, refer to:

- "Using Pluggable Login Modules to perform Java 2 Platform, Enterprise Edition identity to Resource Access Control Facility user mapping" on page 1011
- "Using a Java Authentication and Authorization Services login module to map a registry principal to a System Authorization Facility user ID" on page 1020
- "Updating System Login Configurations to perform a System Authorization Facility identity user mapping" on page 1009

Configuration entry settings for Java Authentication and Authorization Service:

Use this page to specify a list of Java Authentication and Authorization Service (JAAS) login configurations for the application code to use, including J2EE components such as enterprise beans, JavaServer Pages (JSP) files, servlets, resource adapters, and message data blocks (MDBs).

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

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS configuration > Application logins**.

Read the JAAS specifications before you begin defining additional login modules for authenticating to the WebSphere Application Server security run time. You can define additional login configurations for your applications. However, if the WebSphere Application Server LoginModule (`com.ibm.ws.security.common.auth.module.WSLoginModuleImpl`) is not used or the LoginModule does not produce a credential that is recognized by WebSphere Application Server, then the WebSphere Application Server security run time cannot use the authenticated subject from these login configurations for an authorization check for resource access.

Note: You must invoke Java client programs that use Java Authentication and Authorization Service (JAAS) for authentication with a JAAS configuration file specified. The WebSphere product supplies the default JAAS configuration file, wsjaas_client.conf under the *`install_root`*`/properties` directory. This configuration file is set in the */`install_root`*`/bin/launchClient.bat` file as: `set JAAS_LOGIN_CONFIG=-Djava.security.auth.login.config=%WAS_HOME%\properties\wsjaas_client.conf`

If `launchClient.bat` file is not used to invoke Java client programs, make sure that the appropriate JAAS configuration file is passed to the Java virtual machine with the *-Djava.security.auth.login.config* flag.

ClientContainer:

Specifies the login configuration used by the client container application, which uses the CallbackHandler API defined in the client container deployment descriptor.

The ClientContainer configuration is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use it fail.

Default: ClientContainer

DefaultPrincipalMapping:

Specifies the login configuration used by Java 2 Connectors to map users to principals that are defined in the J2C Authentication Data Entries.

ClientContainer is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use it fail.

Default: ClientContainer

WSLogin:

Specifies whether all applications can use the WSLogin configuration to perform authentication for the WebSphere Application Server security run time.

This login configuration does not honor the CallbackHandler defined in the client container deployment descriptor. To use this functionality, use the ClientContainer login configuration.

The WSLogin configuration is the default login configuration for the WebSphere Application Server. Do not remove this default because other administrative applications that use it will fail. This login configuration authenticates users for the WebSphere Application Server security run time. Use credentials from the authenticated subject returned from this login configurations as an authorization check for access to WebSphere Application Server resources.

Default: ClientContainer

System login configuration entry settings for Java Authentication and Authorization Service:

Use this page to specify a list of Java Authentication and Authorization Service (JAAS) system login configurations.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **JAAS configuration > System logins**.

Read the Java Authentication and Authorization Service documentation before you begin defining additional login modules for authenticating to the WebSphere Application Server security run time. Do not remove the following system login modules:

- ICSF
- RMI_INBOUND
- WEB_INBOUND

- DEFAULT
- RMI_OUTBOUND
- SWAM
- SWAM_ZOSMAPPING
- wssecurity.IDAssertion
- wssecurity.signature
- wssecurity.PKCS7
- wssecurity.PkiPath
- wssecurity.UsernameToken
- wssecurity.X509BST
- LTPA
- LTPA_WEB

ICSF:

Processes login requests when Integrated Cryptographic Services Facility (ICSF) is used as the authentication mechanism.

RMI_INBOUND, WEB_INBOUND, DEFAULT:

Process inbound login requests for Remote Method Invocation (RMI), Web applications, and most of the other login protocols. These login configurations are used by WebSphere Application Server Version 5.1.1 and later.

RMI_INBOUND
The RMI_INBOUND login configuration handles logins for inbound RMI requests. Typically, these logins are requests for authenticated access to Enterprise JavaBeans (EJB) files. Also, these logins might be Java Management Extensions (JMX) requests when using the RMI connector.

WEB_INBOUND
The WEB_INBOUND login configuration handles logins for Web application requests, which includes servlets and JavaServer Pages (JSP) files. This login configuration can interact with the output that is generated from a trust association interceptor (TAI), if configured. The Subject passed into the WEB_INBOUND login configuration might contain objects generated by the TAI.

DEFAULT
The DEFAULT login configuration handles the logins for inbound requests made by most of the other protocols and internal authentications.

These three login configurations will pass in the following callback information, which is handled by the login modules within these configurations. These callbacks are not passed in at the same time. However, the combination of these callbacks determines how WebSphere Application Server authenticates the user.

Callback
```
callbacks[0] = new javax.security.auth.callback.
NameCallback("Username: ");
```

Responsibility
Collects the user name that is provided during a login. This information can be the user name for the following types of logins:
- User name and password login, which is known as basic authentication.
- User name only for identity assertion.

Callback
```
callbacks[1] = new javax.security.auth.callback.
PasswordCallback("Password: ", false);
```

Responsibility

Collects the password that is provided during a login.

Callback

```
callbacks[2] = new com.ibm.websphere.security.auth.callback.
WSCredTokenCallbackImpl("Credential Token: ");
```

Responsibility

Collects the Lightweight Third Party Authentication (LTPA) token (or other token type) during a login. Typically, this information is present when a user name and a password are not present.

Callback

```
callbacks[3] = new com.ibm.wsspi.security.auth.callback.
WSTokenHolderCallback("Authz Token List: ");
```

Responsibility

Collects the ArrayList of the TokenHolder objects that are returned from the call to the WSOpaqueTokenHelper. createTokenHolderListFromOpaqueToken () method using the Common Secure Interoperability version 2 (CSIv2) authorization token as input.

Restriction: This callback is present only when the **Security Attribute Propagation** option is enabled and this login is a propagation login. In a propagation login, sufficient security attributes are propagated with the request to prevent having to access the user registry for additional attributes.

In system login configurations, WebSphere Application Server authenticates the user based upon the information collected by the callbacks. However, a custom login module does not need to act upon any of these callbacks. The following list explains the typical combinations of these callbacks:

- The `callbacks[0] = new javax.security.auth.callback.NameCallback("Username: ");` callback only

  This callback occurs for CSIv2 Identity Assertion; Web and CSIv2 X509 certificate logins; old-style trust association interceptor logins, and so on. In Web and CSIv2 X509 certificate logins, WebSphere Application Server maps the certificate to a user name. This callback is used by any login type that establishes trust using the user name only.
- Both the `callbacks[0] = new javax.security.auth.callback.NameCallback("Username: ");` callback and the `callbacks[1] = new javax.security.auth.callback.PasswordCallback("Password: ", false);` callbacks.

  This combination of callbacks is typical for basic authentication logins. Most user authentications occur using these two callbacks.
- The `callbacks[2] = new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl("Credential Token: ");` only

  This callback is used to validate a Lightweight Third Party Authentication (LTPA) token. This validation typically occurs during an single signon (SSO) or downstream login. Any time a request originates from a WebSphere Application Server, instead of a pure client, the LTPA token typically flows to the target server. For single signon (SSO), the LTPA token is received in the cookie and the token is used for login. If a custom login module needs the user name from an LTPA token, the module can use the following method to retrieve the unique ID from the token:

  ```
  com.ibm.wsspi.security.token.WSSecurityPropagationHelper.
  validateLTPAToken(byte[])
  ```

  After retrieving the unique ID, use the following method to get the user name:

  ```
  com.ibm.wsspi.security.token.WSSecurityPropagationHelper.
  getUserFromUniqueID(uniqueID)
  ```

  Important: Any time a custom login module is plugged in ahead of the WebSphere Application Server login modules and it changes the identity using a credential mapping service, it is important

that this login module validates the LTPA token, if present. Calling the following method is sufficient to validate the trust in the LTPA token:

```
com.ibm.wsspi.security.token.WSSecurityPropagationHelper.
validateLTPAToken(byte[])
```

The receiving server must have the same LTPA keys as the sending server in order for this to be successful. There is a possible security exposure if you do not validate this LTPA token, when present.

- A combination of any of the previously mentioned callbacks plus the `callbacks[3] = new com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback("Authz Token List: ");` callback.

  This callback indicates that some propagated attributes arrived at the server. The propagated attributes still require one of the following authentication methods:

  - `callbacks[0] = new javax.security.auth.callback.NameCallback("Username: ");`
  - `callbacks[1] = new javax.security.auth.callback.PasswordCallback("Password: ", false);`
  - `callbacks[2] = new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl("Credential Token: ");`

  If the attributes are added to the Subject from a pure client, then the NameCallback and PasswordCallback callbacks authenticate the information and the objects that are serialized in the token holder are added to the authenticated Subject.

  If both CSIv2 identity assertion and propagation are enabled, WebSphere Application Server uses the NameCallback and the token holder, which contains all of the propagated attributes, to deserialize most of the objects. WebSphere Application Server uses the NameCallback because trust is established with the servers that you indicate in the CSIv2 trusted server list. To specify trusted servers, complete the following steps:

  1. Click **Security > Global security**.
  2. Under Authentication, click **Authentication protocol > CSIv2 Inbound authentication**.

  Custom serialization needs to be handled by a custom login module. For more information, see "Security attribute propagation".

In addition to the callbacks defined previously, the WEB_INBOUND login configuration only can contain the following additional callbacks

Callback

```
callbacks[4] = new com.ibm.websphere.security.auth.callback.
WSServletRequestCallback("HttpServletRequest: ");
```

Responsibility

Collects the HTTP servlet request object, if presented. This callback enables login modules to retrieve information from the HTTP request to use during a login.

Callback

```
callbacks[5] = new com.ibm.websphere.security.auth.callback.
WSServletResponseCallback("HttpServletResponse: ");
```

Responsibility

Collects the HTTP servlet response object, if presented. This callback enables login modules to add information into the HTTP response as a result of the login. For example, login modules might add the SingleSignonCookie to the response.

Callback

```
callbacks[6] = new com.ibm.websphere.security.auth.callback.
WSAppContextCallback("ApplicationContextCallback: ");
```

Responsibility

Collects the Web application context used during the login. This callback consists of a Hashtable, which contains the application name and the redirect Web address, if present.

The following login modules are predefined for the RMI_INBOUND, WEB_INBOUND, and DEFAULT system login configurations. You can add custom login modules before, between, or after any of these login modules, but you cannot remove these predefined login modules.

- com.ibm.ws.security.server.lm.ltpaLoginModule

  This login module performs the primary login when attribute propagation is either enabled or disabled. A primary login uses normal authentication information such as a user ID and password; an LTPA token; or a trust association interceptor (TAI) and a certificate distinguished name (DN). If any of the following scenarios are true, this login module is not used and the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule module performs the primary login:
  - The java.util.Hashtable object with the required user attributes is contained in the Subject.
  - The java.util.Hashtable object with the required user attributes is present in the sharedState HashMap of the LoginContext.
  - The WSTokenHolderCallback callback is present without a specified password. If a user name and a password are present with a WSTokenHolderCallback, callback, which indicates propagated information, the request likely originates from either a pure client or a server from a different realm that mapped the existing identity to a user ID and password.
- com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule

  This login module performs the primary login using the normal authentication information if any of the following conditions are true:
  - A java.util.Hashtable object with required user attributes is contained in the Subject
  - A java.util.Hashtable object with required user attributes is present in the sharedState HashMap of the LoginContext
  - The WSTokenHolderCallback callback is present without a PasswordCallback callback.

  When the java.util.Hashtable object is present, the login module maps the object attributes into a valid Subject. When the WSTokenHolderCallback is present, the login module deserializes the byte token objects and regenerates the serialized Subject contents. The java.util.Hashtable takes precedence over all of the other forms of login. Be careful to avoid duplicating or overriding what WebSphere Application Server might have propagated previously. By specifying a java.util.Hashtable to take precedence over other authentication information, the custom login module must have already verified the LTPA token, if present, to establish sufficient trust. The custom login module can use the com.ibm.wsspi.security.token.WSSecurityPropagationHelper.validationLTPAToken(byte[]) method to validate the LTPA token present in the WSCredTokenCallback. Failure to validate the LTPA token presents a security risk.

  For more information on adding a Hashtable containing well-known and well-formed attributes used by WebSphere Application Server as sufficient login information, see ″Configuring inbound identity mapping″.

RMI_OUTBOUND:

Processes Remote Method Invocation (RMI) requests that are sent outbound to another server when either the com.ibm.CSI.rmiOutboundLoginEnabled or the com.ibm.CSIOutboundPropagationEnabled properties are true.

These properties are set in the CSIv2 authentication panel. To access the panel, click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 outbound authentication**. To set the com.ibm.CSI.rmiOutboundLoginEnabled property, select **Custom outbound mapping**. To set the com.ibm.CSIOutboundPropagationEnabled property, select the **Security attribute propagation** option.

This login configuration determines the security capabilities of the target server and its security domain. For example, if WebSphere Application Server Version 5.1.1 or later (or 5.1.0.2 for z/OS) communicates with a version 5.x Application Server, then the Version 5.1.1 Application Server sends the authentication information only, using an LTPA token, to the Version 5.x Application Server. However, if WebSphere Application Server Version 5.1.1 or later communicates with a version 5.1.x Application Server, the authentication and authorization information is sent to the receiving application server if propagation is enabled at both the sending and receiving servers. When the application server sends both the authentication and authorization information downstream, it removes the need to re-access the user registry and look up the security attributes of the user for authorization purposes. Additionally, any custom objects added at the sending server should be present in the Subject at the downstream server.

The following callback is available to in the RMI_OUTBOUND login configuration. You can use the `com.ibm.wsspi.security.csiv2.CSIv2PerformPolicy` object that is returned by this callback to query the security policy for this particular outbound request. This query can help determine if the target realm is different than the current realm and if WebSphere Application Server must map the realm. For more information, see ″Configuring outbound mapping to a different target realm″.

Callback

```
callbacks[0] = new WSProtocolPolicyCallback("Protocol Policy Callback: ");
```

Responsibility

Provides protocol-specific policy information for the login modules on this outbound invocation. This information is used to determine the level of security, including the target realm, target security requirements, and coalesced security requirements.

The following method obtains the CSIv2PerformPolicy from this specific login module:

```
csiv2PerformPolicy = (CSIv2PerformPolicy)
((WSProtocolPolicyCallback)callbacks[0]).getProtocolPolicy();
```

A different protocol other than RMI might have a different type of policy object.

The following login module is predefined in the RMI_OUTBOUND login configuration. You can add custom login modules before, between, or after any of these login modules, but you cannot remove these predefined login modules.

com.ibm.ws.security.lm.wsMapCSIv2OutboundLoginModule

Retrieves the following tokens and objects before creating an opaque byte that is sent to another server by using the Common Secure Interoperability version 2 (CSIv2) authorization token layer:

- Forwardable com.ibm.wsspi.security.token.Token implementations from the Subject
- Serializable custom objects from the Subject
- Propagation tokens from the thread

You can use a custom login module prior to this login module to perform credential mapping. However, it is recommended that the login module change the contents of the Subject that is passed in during the login phase. If this recommendation is followed, the login modules processed after this login module act on the new Subject contents.

For more information, see ″Configuring outbound mapping to a different target realm″.

SWAM:

Processes login requests in a single server environment when Simple WebSphere Authentication Mechanism (SWAM) is used as the authentication method.

SWAM does not support forwardable credentials. When SWAM is the authentication method, WebSphere Application Server cannot send requests from server to server. In this case, you must use LTPA.

SWAM_ZOSMAPPING:

This login configuration enables you to map an ID in an Lightweight Directory Access Protocol (LDAP) user registry to a System Authorization Facility (SAF) user ID

wssecurity.IDAssertion:

Processes login configuration requests for Web services security using identity assertion. This login configuration is for version 5.x systems.

wssecurity.PKCS7: This login configuration is for version 6.x systems.

wssecurity.PkiPath: This login configuration is for version 6.x systems.

wssecurity.signature:

Processes login configuration requests for Web services security using digital signature validation. This login configuration is for version 5.x systems.

wssecurity.UsernameToken: This login configuration is for version 6.x systems.

wssecurity.X509BST: This login configuration is for version 6.x systems.

LTPA_WEB:

Processes login requests to components in the Web container such as servlets and JavaServer pages (JSP) files.

The com.ibm.ws.security.web.AuthenLoginModule login module is predefined in the LTPA login configuration. You can add custom login modules before or after this module in the LTPA_WEB login configuration.

The LTPA_WEB login configuration can process the HttpServletRequest object, the HttpServletResponse object, and the Web application name that are passed in using a callback handler. For more information, see ″Customizing a server-side Java Authentication and Authorization Service authentication and login configuration″ in the documentation.

LTPA:

Processes login requests that are not handled by the LTPA_WEB login configuration.

This login configuration is used by WebSphere Application Server Version 5.1 and previous versions.

The com.ibm.ws.security.server.lm.ltpaLoginModule login module is predefined in the LTPA login configuration. You can add custom login modules before or after this module in the LTPA login configuration. For more information, see ″Customizing a server-side Java Authentication and Authorization Service authentication and login configuration″ in the documentation.

Login module settings for Java Authentication and Authorization Service:

Use this page to define the login module for a Java Authentication and Authorization Service (JAAS) login configuration.

You can define the JAAS login modules for application and system logins. To define these login modules in the administrative console, use one of the following paths:

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

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS configuration > Application logins** or **System logins** > *alias_name*.
3. Under Additional properties, click **JAAS login modules**.

Module class name:

Specifies the class name of the given login module.

Data type: String

Proxy class name:

Specifies the name of the proxy login module class.

The default login modules defined by the WebSphere product use the proxy LoginModule class, com.ibm.ws.security.common.auth.module.WSLoginModuleProxy. This proxy class loads the WebSphere Application Server login module with the thread context class loader and delegates all the operations to the *real* login module implementation. The real login module implementation is specified as the delegate option in the option configuration. The proxy class is needed because the Developer Kit application class loaders do not have visibility of the WebSphere Application Server product class loaders.

Data type: String

Authentication Strategy:

Specifies the authentication behavior as authentication proceeds down the list of login modules.

A Java Authentication and Authorization Service (JAAS) authentication provider supplies the authentication strategy. In JAAS, an authentication strategy is implemented through the LoginModule interface.

Data type: String
Default: Required
Range: Required, Requisite, Sufficient and Optional

Required
The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list for each realm.

Requisite
The LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list in the realm entry. If it fails, control immediately returns to the application--that is, authentication does not proceed down the LoginModule list.

Sufficient
The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application--again, authentication does not proceed down the LoginModule list. If it fails, authentication continues down the list.

Optional
The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

Specify additional options by clicking **Custom Properties** under Additional Properties. These name and value pairs are passed to the login modules during initialization. This process is one of the mechanisms that is used to passed information to login modules.

Module order:

Specifies the order in which the Java Authentication and Authorization Service (JAAS) login modules are processed.

Click **Set Order** to change the processing order of the login modules.

Login module order settings for Java Authentication and Authorization Service:

Use this page to specify the order in which WebSphere Application Server processes the login configuration modules.

You can specify the order of the login modules for application and system logins. To define these login modules in the administrative console, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS Configuration > Application logins** or **System logins** > *login_configuration*. You can create a new configuration by clicking **New**.
3. Under Additional properties, click **JAAS login modules**.
4. Click **Set order**.

When you select one of the JAAS login module class names, you can move that class name up and down the list. After you press **OK** and save the changes, the new order is reflected on either the Application login configuration or System login configuration panel.

Login configuration settings for Java Authentication and Authorization Service:

Use this page to configure application login configurations.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **JAAS configuration > Application logins** or **System logins** > *alias_name*.

Click **Apply** to save changes and to add the extra node name that precedes the original alias name. Clicking **OK** does not save the new changes in the `security.xml` file.

Alias:

Specifies the alias name of the application login.

Do not use the forward slash character (/) in the alias name when defining JAAS login configuration entries. The JAAS login configuration parser cannot process the forward slash character.

Data type: String

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

Applications define whether to use application-managed sign-on or container-managed sign-on 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 should perform sign-on or whether it should enable the WebSphere Application Server to sign-on to the resource manager using the principal mapping configuration. The resource-ref element is defined at application assembly time. Use the WebSphere Development Toolkit to configure the resource `-ref`.

Application managed sign-on

To access an EIS system, applications locate a connection factory from the 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 getConnection, 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 de-compile 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 allows a component-managed authentication alias to be specified on a resource. This authentication data is common to all references to the resource. On the **Resource Adapter**>**Connection Factory** configuration panel, select **component-managed authentication alias**.

With res-auth=Application, the authentication data is taken from, in order:

1. user id and password passed to getConnection(...)
2. component-managed auth alias on the Connection Factory or DataSource
3. Custom Properties UserName and Password on the DataSource

The username and password properties can be initially defined in the RAR file, and can also be defined in the admininstrative console or wsadmin scripting under custom properties. Do not use the custom properties, which enable users to connect to the resources.

Container-managed sign-on

The user ID and password for the target EIS can be supplied by the application server. WebSphere Application Server provides container-managed sign-on functionality. It 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 sign-on, 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(s) to map a client security identity and credentials on the thread of execution to a pre-configured user ID and password.

WebSphere Application Server ships a default many-to-one credential mapping LoginModule that maps any client identity on the thread of execution to a pre-configured user ID and password for a specified target EIS. The default mapping module is a special purpose JAAS LoginModule that returns a PasswordCredential specified by the configured J2C authentication data entry. The default mapping LoginModule 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 Global Security panel under **Authentication** > **JAAS Configuration**. The default principal and credential mapping function is defined by the DefaultPrincipalMapping application JAAS login configuration.

The DefaultPrincipalMapping login configuration should not be modified since 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, or stacking a custom LoginModule in the configuration.

On the z/OS platform, if no user ID and password are present (or defaulted by an alias), WebSphere Application Server runtime searches for a System Authorization Facility (SAF) user ID in the subject (if one is present).

For most systems, the default configuration 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, you must configure the mapping configuration using the **Map resource references to resources** link on an enterprise application panel.

Map resource references to references

To map resource references to resources, do the following:

1. Click **Applications** > **Enterprise Applications**.
2. Select an application.
3. Under Additional Properties, select **Map resource references to resources**.
4. Select a connection factory reference binding from the table that has a login configuration of Resource authorization: Container. You must specify an authentication method for the selected connection factory reference binding. Choose either **Use default method** or **Use custom login configuration**. If you choose the **Use default method** option, the WebSphere Application Server DefaultPrincipalMapping login configuration is selected. You must select an authentication data alias from the drop-down list.
5. After you make a selection, click **Apply** for the configuration to take effect.
6. If you choose **Use custom login configuration**, you must select a mapping JAAS login configuration from the drop-down list.
7. Click **Apply**. The selected login configuration name and an **Update** button appear in the login configuration field of the particular connection factory reference binding.
8. Click **Update** to define mapping properties that you might need to pass to the mapping LoginModule(s).

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 Version 6, 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, (such the as Tivoli Access Manager Global Sign-On (GSO) service). You have two EIS servers: DB2 and MQ.

Use the Tivoli Access Manager GSO to locate authentication data for both backend servers. 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 TAM GSO service. Use the mapping properties in a connection factory reference binding to specify which EIS server the user ID and password are required for.

For more detailed information about developing a mapping module, see the Developing your own Java 2 security mapping module article.

Note:

- WebSphere Application Server Version 6 configures container-managed sign-on under each enterprise application. This is different than WebSphere Application Server Version 5, which configures container-managed sign-on for each connection factory.
- The deprecated way of configuration at the **Resource Adapter** > **Connection factory** panel still works in WebSphere Application Server Version 6. The advantage to configuring at the connection factory reference level is that the configuration has application scope and is not visible to other applications. However, the mapping configuration defined at the connection factory is visible to other applications.
- The mapping configuration at the connection factory has moved to the resource manager connection factory reference. The mapping LoginModules that were developed using WebSphere Application Server Version 5 JAAS Callback types can be used by the resource manager connection factory reference, but the mapping LoginModules cannot take advantage of the custom mapping properties feature.
- Connection factory reference binding supports mapping properties, and passes those properties to mapping LoginModules by way of a new WSMappingPropertiesCallback Callback type. In addition, WSMappingPropertiesCallback and the new WSManagedConnectionFactoryCallback are defined in the com.ibm.wsspi package. New mapping LoginModules should use the new Callback types.

Managing J2EE Connector Architecture authentication data entries:

Java 2 Platform, Enterprise Edition (J2EE) Connector authentication data entries are used by resource adapters and Java DataBase Connectivity (JDBC) data sources. A J2EE Connector authentication data entry contains authentication data, which includes the following information:

Alias An identifier that identifies the authentication data entry. When configuring resource adapters or data sources, the administrator can specify which authentication data to choose using the corresponding alias.

User ID
A user identity of the intended security domain. For example, if a particular authentication data entry is used to open a new connection to DB2, this entry contains a DB2 user identity.

Password
The password of the user identity is encoded in the configuration respository.

Description
A short text description.

This task creates and deletes Java 2 Connector (J2C) authentication data entries.

1. Delete a J2C authentication data entry.
   a. Click **Security > Global security**.
   b. Under Authentication, click **JAAS configuration > J2C authentication data**. The **J2C Authentication Data Entries** panel is displayed.
   c. Select the check boxes for the entries to delete and click **Delete**. Before deleting or removing an authentication data entry, make sure that it is not used or referenced by any resource adapter or data source. If the deleted authentication data entry is used or referenced by a resource, the application that uses the resource adapter or the data source fails to connect to the resources.
2. Create a new J2C authentication data entry.
   a. Click **Security > Global security**. The **J2C Authentication Data Entries** panel is displayed.
   b. Under Authentication, click **JAAS configuration > J2C authentication data**. The **J2C Authentication Data Entries** panel is displayed.
   c. Click **New**.
   d. Enter a unique alias, a valid user ID, a valid password, and a short description (optional).
   e. Click **OK** or **Apply**. No validation for the user ID and password is required.
   f. Click **Save**. For a Network Deployment installation, make sure that a file synchronized operation is performed to propagate the changes to other nodes.

A new J2C authentication data entry is created or an old entry is removed. The newly created entry is visible without restarting the application server process to use in the data source definition. But the entry is only in effect after the server is restarted. Specifically, the authentication data is loaded by an application server when starting an application and is shared among applications in the same application server.

If you create or update a data source that points to a newly created J2C authentication data alias, the test connection fails to connect until you restart the deployment manager. After you restart the deployment manager, the J2C authentication data is reflected in the run-time configuration. Any changes to the J2C authentication data fields require a deployment manager restart for the changes to take effect.

This step defines authentication data that you can share among resource adapters and data sources. Use the authentication data entry that is defined in the resource adapters or the data sources.

Java 2 Connector authentication data entry settings:

Use this page as a central place for administrators to define authentication data, which includes user identities and passwords. These values can reference authentication data entries by resource adapters, data sources, and other configurations that require authentication data using an alias.

You can display this page directly from the JAAS configuration page or from other pages for resources that use J2EE Connector (J2C) authentication data entries. For example, to view this administrative page, click **Security > Global security**. Under Authentication, click **JAAS configuration > J2C authentication data**.

Deleting authentication data entries: Be careful when deleting authentication data entries. If the deleted authentication data is used by other configurations, the initializing resources process fails.

Define a new authentication data entry by clicking **New**.

Alias:

Specifies the name of the authentication data entry.

Data type:	String
Units:	String

Default: None

User ID:

Specifies the user identity.

Data type: String

Password:

Specifies the password that is associated with the user identity.

Data type: String

Description:

Specifies an optional description of the authentication data entry. For example, this authentication data entry is used to connect to DB2.

Data type: String

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 as-is. 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, it 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 the 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 1.3 and higher compliant application servers. When using identity assertion, 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 1158.

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 can not be determined at the inbound server to do inbound mapping. For more information, see “Configuring outbound mapping to a different target realm” on page 1047.

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 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.” In other cases, such as identity assertion, you might receive a user name in the NameCallback of the inbound login configuration that enables you to map the identity.

Configuring inbound identity mapping

For inbound identity mapping, it is recommend that you 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 of the following callbacks present: NameCallback and the WSCredTokenCallback. 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 except if invalid authentication data exists.
    // 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 login 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 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 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 WSPrincipal objects 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 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 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)
    {
     // Retrives 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 look up 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 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 sharedState 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 sharedState 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 ArrayList 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 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 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 ArrayList

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 cacheKey needs to include location A so that the Subject 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 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 > Global security**.
   b. Under Authentication, click **JAAS configuration > System logins > RMI_INBOUND**
   c. Under Additional Properties, click **JAAS login modules > New** to add your login module to the RMI_INBOUND configuration.
   d. Return to the JAAS login modules panel for RMI_INBOUND and click **Set order** to change the order that the login modules are loaded so that WebSphere Application Server loads your custom login module first.
   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” article shows a custom login module that creates a java.util.Hashtable based on the specified NameCallback. The java.util.Hashtable is added to the sharedState java.util.Map so that the WebSphere Application Server login modules can locate the information in the Hashtable.

Example: Custom login module for inbound mapping:

This sample shows a custom login module that creates a java.util.Hashtable hastable that is based on the specified NameCallback callback. The java.util.Hashtable hash table is added to the sharedState java.util.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)
{
 // Invalid authentication data. 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 login for you. When passwords are presented, be very careful not to
    // 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 is changed, clear the propagated attributes below
 // 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 look up mechanism for
  // the created Subject. The cache key can be an object, but should have
  // 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 sharedState 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 1052.
  // 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.)

 // Not doing anything here for this specific example
}

// 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 request is rejected to protect against a rogue server reading potentially sensitive information if successfully impersonating the home of the object. The following alternatives are available to enable one server to send outbound requests to a target server in a different realm:

- 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. To allow information to flow to a server in a different realm, complete the following steps in the administrative console:
  1. Click **Security > Global security**.
  2. Under Authentication, click **Authentication protocol > 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 OS. 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 then 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 1048.
  2. Use the code sample in “Example: Using the WSLogin configuration to create a basic authentication subject” on page 1048 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 > Global security**.
     b. Under Authentication, click **Authentication protocol > 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 1049 article shows a custom login module that determines whether the realm names match. In this example, the realm names do not match so the WSLogin 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 > Global security**.

2) Under Authentication, click **JAAS configuration > 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 and 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 > Global security**.
  2. Under Authentication, click **JAAS Configuration > 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.
  9. Click **OK**.

  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 login 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 is an alternative that  validates the user ID and
```

```
  // password specified against the target realm. It will perform 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 invalid. If false is
  // returned, a WSLoginFailedException is thrown. 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();
   }
  });

  // Executes 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 Javadoc for more information.
  csiv2PerformPolicy = (CSIv2PerformPolicy) ((WSProtocolPolicyCallback)callbacks[0]).
        getProtocolPolicy();

  // Checks if the realms do not match. If they do not match, then login 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;
}
```

Security attribute propagation

Security attribute propagation enables WebSphere Application Server to 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 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 de-serialize 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 run time 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 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 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 an Lightweight Third Party Authentication (LTPA) token, and then deserializing a series of tokens that constitute both custom objects and token framework objects known to the 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 PropagationToken” on page 1058 and “Implementing a custom PropagationToken” on page 1064.

Single signon (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 together with the getVersion method defines the cookie name. WebSphere Application Server defines a default SingleSignonToken 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 signon scenario. For more information on SSO tokens, see “Configuring single signon” on page 931. For information on how to use and implement this token type, see “Default SingleSignonToken” on page 1085 and “Implementing a custom SingleSignonToken” on page 1086.

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 executed by the target server. The attributes 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 PropagationToken” on page 1058 and “Implementing a custom PropagationToken” on page 1064.

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 do 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 the purpose of the custom data to use the token to perform custom authentication decisions. For information on how to use and implement this token type, see “Default AuthenticationToken” on page 1097 and “Implementing a custom AuthenticationToken” on page 1098.

Horizontal propagation versus downstream propagation

In WebSphere Application Server, both horizontal propagation, which is uses single signon 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 amongst 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 signon (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. In order to enable horizontal propagation, you must configure the single signon token and the Web inbound security attribute propagation features. You can configure both of these features using the administrative console by completing the following steps:

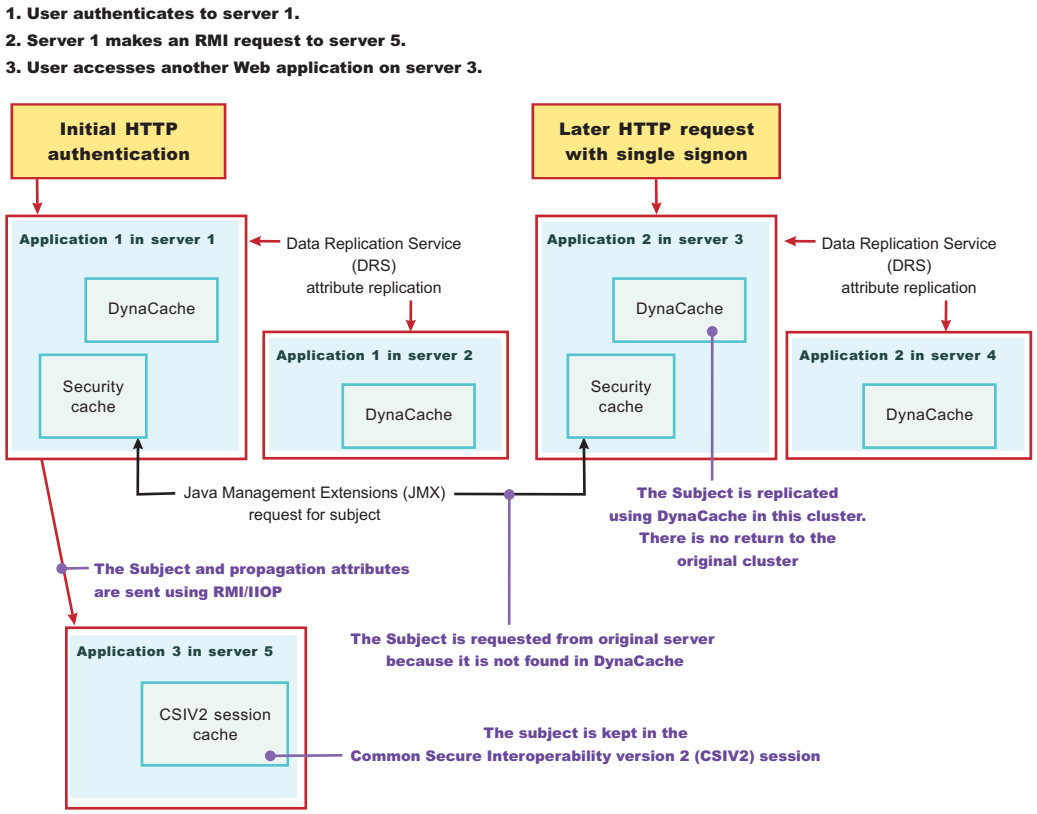
1. Click **Security > Global security**.
2. Under Authentication, click **Authentication Mechanisms > LTPA**.
3. Under Additional properties, click **Single signon (SSO)**

For more information, see “Enabling security attribute propagation” on page 1056.

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 5 to 6 separate remote calls.

Figure 1



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.

Enabling security attribute propagation

The security attribute propagation feature of WebSphere Application Server enables you to send security attribute information regarding the original login to other servers using a token. To fully enable security attribute propagation, you must configure the single signon (SSO), 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**. Under Additional properties, click **Server-level 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*:9060/ibm/console The administrative console address might differ if you have previously changed the port number.
2. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**. Under Additional Properties, click **Single Signon (SSO)**.
3. **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 PropagationToken, but ignore the security attribute information that it does not understand.
4. 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. This attributes include any custom attributes that might be set in the PropagationToken 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 measurement the performance of your environment 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.
5. Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**. The Login configuration field specifies `RMI_INBOUND` as the system login configuration used for inbound requests. To add custom Java Authentication and Authorization Service (JAAS) login modules, complete the following steps:
   a. Click **Security > Global security**. Under Authentication, click **JAAS Configuration > 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.
   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 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 because this prefix is reserved for WebSphere Application Server internal use.

      You can specify the order in which the login modules are processed by clicking **Set Order**.
6. 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.
7. Click **Security**. Under Authentication, click **Authentication protocol > CSIv2 Outbound authentication**. The CSIv2 outbound authentication panel is displayed. The **Login configuration** field specifies `RMI_OUTBOUND` as 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 listed previously for CSIv2 Inbound authentication.
8. **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 Common Secure Interoperability version 2 (CSIv2) protocol to send the Subject and PropagationToken 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.

Important: When using security attribute propagation, use the same LTPA keys in all cell configurations.

9. **Optional:** Select the **Custom Outbound Mapping** option if you deselect the **Security Attribute Propagation** option and you want to use the RMI_OUTBOUND login configuration. If 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.
10. **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 allow for inbound mapping to occur 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 the request being rejected due to 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.
11. **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
    ```

After completing these steps, you have configured WebSphere Application Server to propagate security attributes to other servers. After you have configured WebSphere Application Server for security attribute propagation and need to disable this functionality, you can disable propagation for either the server level or the cell level. To disable security attribute propagation on the server level, click **Server > Application Servers >***server_name*. Under Security, click **Server security**. You can disable security attribute propagation for inbound requests by clicking **CSI inbound authentication** under Additional Properties and deselecting **Security attribute propagation**. You can disable security attribute propagation for outbound requests by clicking **CSI outbound authentication** under Additional Properties and deselecting **Security attribute propagation**. 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.

Default PropagationToken

A default PropagationToken is located on the thread of execution for applications and the security infrastructure to use. WebSphere Application Server propagates this default PropagationToken downstream and the token stays on the thread where the invocation lands at each hop. The data should be available from within the container of any resource where the PropagationToken lands. Remember that you must enable the propagation feature at each server where a request is sent in order for propagation to work. Make sure that you have enabled 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 article documents the usage scenarios and includes examples. A close relationship exists between PropagationToken and the WorkArea feature. The main difference between these features is that after you add attributes to the PropagationToken, you cannot change the attributes. You cannot change these attributes so that the security run time 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 is stored to hold that attribute. Any new attribute added with the same key is added to the ArrayList. When you call getAttributes, the ArrayList is converted to a String[] and the order is preserved. The first element in the String[] is the first attribute added for that specific key.

In the default PropagationToken, 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 re-send the authentication information downstream 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 PropagationToken changes, a new session is generated and subsequently a new authentication occurs. Frequent changes to the PropagationToken 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 PropagationToken

Every time the PropagationToken is propagated and used to create the authenticated Subject, either horizontally or downstream, the name of the receiving application server is logged into the PropagationToken. 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*:*server*. The output, for example, is:
`myManager:node1:server1`

Getting the caller list from the default PropagationToken

A default PropagationToken is generated any time an authenticated user is set on the thread of execution or any one tries to add attributes to the PropagationToken. Whenever an authenticated user is set on the thread, the user is logged in the default PropagationToken. There may be some pushing and popping of Subjects by the authorization code. 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 added to the thread gets logged into the PropagationToken:

- The current Subject must be authenticated. For example, an unauthenticated Subject is not logged.
- The current authenticated Subject is logged if a Subject has not been previously logged.
- The current authenticated Subject is logged if the last authenticated Subject logged does not contain the same user.
- The current authenticated Subject is logged on each unique application server 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*:*server*:*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 PropagationToken

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 securityName of the caller only. If you need to know more than the securityName, call the getCallerList() method and retrieve the first entry in the String[]. This entry provides the entire caller information. The following code sample retrieves the securityName 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 PropagationToken

Whenever you want to know what the first application server is for this request, you can 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 PropagationToken

You can add custom attributes to the default PropagationToken for application usage. This token follows the request downstream so that the attributes are available when they are needed. When you use the default PropagationToken to add attributes, you must understand the following issues:

- When you add information to the PropagationToken, it 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 your need. However, all of the values must be available from a returned String[] in the order they were added.
- The PropagationToken is available only on servers where propagation and security are enabled.
- The Java 2 Security javax.security.auth.AuthPermission wssecurity.addPropagationAttribute is needed to add attributes to the default PropagationToken.
- 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 PropagationToken” to retrieve attributes using the getPropagationAttributes application programming interface (API).

Getting custom attributes from the default PropagationToken

Custom attributes are added to the default PropagationToken using the addPropagationAttribute API. These attributes can be retrieved using the getPropagationAttributes API. This token follows the request downstream so the attributes are available when they are needed. When you use the default PropagationToken to retrieve attributes, you must understand the following issues.

- The PropagationToken is available only on servers where propagation and security are enabled.
- The Java 2 Security javax.security.auth.AuthPermission wssecurity.getPropagationAttributes is needed to retrieve attributes from the default PropagationToken.

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 TokenFactory associated with the default PropagationToken

When WebSphere Application Server generates a default PropagationToken, 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 > Global Security**.
2. Under Additional properties, click **Custom properties**.

The default TokenFactory specified for this property is called com.ibm.ws.security.ltpa.AuthzPropTokenFactory. This token factory encodes the data in the PropagationToken and does not encrypt the data. Because the PropagationToken typically flows over Common Secure Interoperability version 2 (CSIv2) using Secure Sockets Layer (SSL), there is no need to encrypt the token itself. However, if you need additional security for the PropagationToken, you can associate a different TokenFactory implementation with this property to get encryption. For example, if you choose to associate com.ibm.ws.security.ltpa.LTPAToken2Factory 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 PropagationToken is a good reason to change the TokenFactory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default PropagationToken, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory implementation instantiates and validates your token implementation. You can choose to use the Lightweight Third Party Authentication (LTPA) keys passed into the initialize method of the

TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default PropagationToken, using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
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 TokenFactory implementation.
4. Verify that your implementation classes are put into the *install directory*/classes directory so that the WebSphere class loader can load the classes.

Implementing a custom PropagationToken

This task explains how you might create your own PropagationToken implementation, which is set on the thread of execution and propagated downstream. The default PropagationToken usually is sufficient for propagating attributes that are not user-specific. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread by plugging in a custom login module into the inbound system login configurations. This task also might include encryption and decryption.

To implement a custom Propagation token, you must complete the following steps:

1. Write a custom implementation of the PropagationToken interface. There are many different methods for implementing the PropagationToken interface. However, make sure that the methods required by the PropagationToken interface and the token interface are fully implemented. After you implement this interface, you can place its classes in the *install_dir*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so that it has the necessary permissions that are needed by the server code.

   Tip: All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the PropagationToken, might extend the abstract class and then most of the work is completed.

   To see an implementation of PropagationToken, see “Example: com.ibm.wsspi.security.token.PropagationToken implementation” on page 1065
2. Add and receive the custom PropagationToken during WebSphere Application Server logins This task is typically accomplished by adding a custom login module to the various application and system login configurations. You also can add the implementation from an application. However, in order to deserialize the information, you will need to plug in a custom login module, which is discussed in “Propagating a custom Java serializable object” on page 1107. The WSSecurityPropagationHelper class has APIs that are used to set a PropagationToken on the thread and to retrieve it from the thread to make updates.

   The code sample in “Example: custom PropagationToken login module” on page 1070 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom PropagationToken implementation and set it on the thread. If

the callback contains propagation data, look for your specific custom PropagationToken TokenHolder instance, convert the byte[] back into your customer PropagationToken object, and set it back on the thread. The code sample shows both instances.

You can add attributes any time your custom PropagationToken is added to the thread. If you add attributes between requests and the getUniqueId method changes, then the CSIv2 client session is invalidated so that it can send the new information downstream. Keep in mind that adding attributes between requests can affect performance. In many cases, this is the desired behavior so that downstream requests receive the new PropagationToken information.

To add the custom PropagationToken to the thread, call WSSecurityPropagationHelper.addPropagationToken. This call requires the following Java 2 Security permission: WebSphereRuntimePerMission ″setPropagationToken″

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom propagation token Also, you can add this login module to any of the application logins where you might want to generate your custom PropagationToken on the thread during the login. Alternatively, you can generate the custom PropagationToken implementation from within your application. However, to deserialize it, you need to add the implementation to the system login modules.

   For information on how to add your custom login module to the existing login configurations, see Custom login module development for a system login configuration

After completing these steps, you have implemented a custom PropagationToken.

Example: com.ibm.wsspi.security.token.PropagationToken implementation:

Use this file to see an example of a PropagationToken implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.PropagationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom PropagationToken, see “Implementing a custom PropagationToken” on page 1064.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomPropagationTokenImpl implements com.ibm.wsspi.security.
   token.PropagationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
   // 2 hours in millis, by default
  private static long expire_period_in_millis = 2*60*60*1000;
 private long counter = 0;

/**
```

```
 * The constructor that is used to create initial PropagationToken instance
 */

 public CustomAbstractTokenImpl ()
 {
  // set the token version
  addAttribute("version", "1");
  // set the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * The constructor that is used to deserialize the token bytes received
 * during a propagation login.
 */
 public CustomAbstractTokenImpl (byte[] token_bytes)
 {
  try
  {
       hashtable = (java.util.Hashtable) com.ibm.wsspi.security.token.
         WSOpaqueTokenHelper.deserialize(token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long type.
 * @return long
 */
 public long getExpiration()
 {
  // get the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // expiration is the first element (should only be one)
```

```
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
     // You can choose whether your token gets propagated. In some cases
     // you might want the token to be local only.
  return true;
 }

/**
 * Gets the principal that this token belongs to. If this token is an
 * authorization token, this principal string must match the authentication
 * token principal string or the message is rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // It is not necessary for the PropagtionToken to return a principal,
     // because it is not user-centric.
  return "";
 }

/**
 * Returns the unique identifier of the token based upon information that
 * the provider considers makes it a unique token. This identifier is used
 * for caching purposes and might be used in combination with other token
 * unique IDs that are part of the same Subject.
 *
 * This method should return null if you want the accessID of the user to
 * represent its uniqueness. This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
     // If you want to propagate the changes to this token, change the
     // value that this unique ID returns whenever the token is changed.
     // Otherwise, CSIv2 uses an existing session when everything else is
     // the same. This getUniqueID ischecked by CSIv2 to determine the
     // session lookup.
  return counter;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
     // Do this if the object is set to read-only during login commit
     // because this guarantees that no new data is set.
    if (isReadOnly() && tokenBytes == null)
```

```
     tokenBytes = com.ibm.wsspi.security.token.WSOpaqueTokenHelper.
               serialize(hashtable);

     // You can deserialize this in the downstream login module using
         //  WSOpaqueTokenHelper.deserialize()
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token, which is used to identify the byte[] in the
 * protocol message.
 * @return String
 */
 public String getName()
 {
  return this.getClass().getName();
 }

/**
 * Gets the version of the token as an short type. This code also is used
 * to identify the byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only. The implementation
 * needs to ensure that any setter methods check that this read-only flag has
 * been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is readonly
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }
```

```
/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name and value pair. Returns the previous values set
 * for the key, or returns null if the value is not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Increments the counter to change the uniqueID
   counter++;

   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all of the attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }
```

```
/**
 * Returns a deep clone of this token. This is typically used by the session
 * logic of the CSIv2 server to create a copy of the token as it exists in the
 * session.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomPropagationTokenImpl deep_clone =
   new com.ibm.websphere.security.token.CustomPropagationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

Example: custom PropagationToken login module:

This file shows how to determine if the login is an initial login or a propagation login

```
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.)

  // Handles the WSTokenHolderCallback to see if this is an initial
     // or propagation login.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // handle exception
  }

  // Receives the ArrayList of TokenHolder objects (the serialized tokens)
```

```
 List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

 if (authzTokenList != null)
 {
  // Iterates through the list looking for your custom token
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

   // Looks for the name and version of your custom PropagationToken implementation
   if (tokenHolder.getName().equals("
             com.ibm.websphere.security.token.CustomPropagationTokenImpl") &&
       tokenHolder.getVersion() == 1)
   {
    // Passes the bytes into your custom PropagationToken constructor
           //  to deserialize
    customPropToken = new
     com.ibm.websphere.security.token.CustomPropagationTokenImpl(tokenHolder.
                 getBytes());

   }
  }
 }
 else // This is not a propagation login. Create a new instance of
         // your PropagationToken implementation
 {
  // Adds a new custom propagation token. This is an initial login
  customPropToken = new com.ibm.websphere.security.token.CustomPropagationTokenImpl();

  // Adds any initial attributes
  if (customPropToken != null)
  {
   customPropToken.addAttribute("key1", "value1");
   customPropToken.addAttribute("key1", "value2");
   customPropToken.addAttribute("key2", "value1");
   customPropToken.addAttribute("key3", "something different");
  }
 }

 // Note: You can add the token to the thread during commit in case
    // something happens during the login.
}

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 (customPropToken != null)
 {
  // Sets the propagation token on the thread
  try
  {

   System.out.println(tc, "*** ADDED MY CUSTOM PROPAGATION TOKEN TO THE THREAD ***");
   // Prints out the values in the deserialized propagation token
   java.util.Enumeration keys = customPropToken.getAttributeNames();
```

```
    while (keys.hasMoreElements())
    {
     String key = (String) keys.nextElement();
     String[] list = (String[]) customPropToken.getAttributes(key);
     for (int k=0; k<list.length; k++)
     System.out.println("Key/Value: " + key + "/" + list[k]);
    }

    // This sets it on the thread using getName() + getVersion() as the key
    com.ibm.wsspi.security.token.WSSecurityPropagationHelper.addPropagationToken(
              customPropToken);
   }
   catch (Exception e)
   {
    // Handles exception
   }


   // Now you can verify that you have set it properly by trying to get
       // it back from the thread and print the values.
   try
   {
    // This gets the PropagationToken from the thread using getName()
          // and getVersion() parameters.
    com.ibm.wsspi.security.token.PropagationToken tempPropagationToken =
     com.ibm.wsspi.security.token.WSSecurityPropagationHelper.getPropagationToken
      ("com.ibm.websphere.security.token.CustomPropagationTokenImpl", 1);

    if  (tempPropagationToken != null)
    {
     System.out.println(tc, "*** RECEIVED MY CUSTOM PROPAGATION
               TOKEN FROM THE THREAD ***");
     // Prints out the values in the deserialized propagation token
     java.util.Enumeration keys = tempPropagationToken.getAttributeNames();
     while (keys.hasMoreElements())
     {
      String key = (String) keys.nextElement();
      String[] list = (String[]) tempPropagationToken.getAttributes(key);
      for (int k=0; k<list.length; k++)
      System.out.println("Key/Value: " + key + "/" + list[k]);
     }
    }
   }
   catch (Exception e)
   {
    // Handles exception
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.PropagationToken customPropToken = null;

}
```

Default AuthorizationToken

This article explains how WebSphere Application Server uses the default AuthorizationToken. Consider using the default AuthorizationToken when you are looking for a place to add string attributes that will get propagated downstream. However, make sure that the attributes that you add to the AuthorizationToken are specific to the user associated with the authenticated Subject. If they are not specific to a user, the attributes probably belong in the PropagationToken, which is also propagated with the request. For more information on the PropagationToken, see “Default PropagationToken” on page 1058. To add attributes into the AuthorizationToken, 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 AuthorizationToken is created in the wsMapDefaultInboundLoginModule after the login occurs in the ltpaLoginModule. A reference to the default AuthorizationToken can be obtained 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 AuthorizationToken..

For more information on the Java Authentication and Authorization Service (JAAS) programming model, see “Security: Resources for learning” on page 879.

Important: Whenever you plug in 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 *install_dir*/classes directory, it has Java 2 Security AllPermissions. 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 needed to execute the application programming interfaces (API) 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 AuthorizationToken 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 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 TokenFactory associated with the default AuthorizationToken

When WebSphere Application Server generates a default AuthorizationToken, 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 > Global Security**.

2. Under Additional properties, click **Custom properties**.

The default TokenFactory that used is called com.ibm.ws.security.ltpa.AuthzPropTokenFactory. This token factory encodes the data, but does not encrypt the data in the AuthorizationToken. Because the AuthorizationToken typically flows over Common Secure Interoperability version 2 (CSIv2) using Secure Sockets Layer (SSL), there is no need to encrypt the token itself. However, if you need addition security for the AuthorizationToken, you can associate a different TokenFactory implementation with this property to get encryption. For example, if you associate com.ibm.ws.security.ltpa.LTPAToken2Factory with this property, the token uses AES encryption. However, you need to weigh the performance impacts against your security needs. Adding sensitive information to the AuthorizationToken is one reason to change the TokenFactory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default AuthorizationToken you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory 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 TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default AuthorizationToken, using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
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 TokenFactory implementation.
4. Verify that your implementation classes are put into the *install directory*/classes directory so that the WebSphere class loader can load the classes.

Implementing a custom AuthorizationToken

This task explains how you might create your own AuthorizationToken implementation, which is set in the login Subject and propagated downstream. The default AuthorizationToken usually is sufficient for propagating attributes that are user-specific. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.
- Affect the overall uniqueness of the Subject using the getUniqueID() application programming interface (API).

To implement a custom authorization token, you must complete the following steps:

1. Write a custom implementation of the AuthorizationToken interface. There are many different methods for implementing the AuthorizationToken interface. However, make sure that the methods required by the AuthorizationToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the *install_dir*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so that it has the necessary permissions that are needed by the server code.

Tip: All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the AuthorizationToken, might extend the abstract class and then most of the work is completed.

To see an implementation of AuthorizationToken, see "Example: com.ibm.wsspi.security.token.AuthorizationToken implementation"

2. Add and receive the custom AuthorizationToken during WebSphere Application Server logins This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, in order to deserialize the information, you must plug in a custom login module, which is discussed in "Propagating a custom Java serializable object" on page 1107. After the object is instantiated in the login module, you can add the object to the Subject during the commit() method.

   If you only want to add information to the Subject to get propagated, see "Propagating a custom Java serializable object" on page 1107. If you want to ensure that the information is propagated, want to do you own custom serialization, or want to specify the uniqueness for Subject caching purposes, then consider writing your own AuthorizationToken implementation.

   The code sample in "Example: custom AuthorizationToken login module" on page 1082 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom AuthorizationToken implementation and set it into the Subject. If the callback contains propagation data, look for your specific custom AuthorizationToken TokenHolder instance, convert the byte[] back into your custom AuthorizationToken object, and set it back into the Subject. The code sample shows both instances.

   You can make your AuthorizationToken read-only in the commit phase of the login module. If you do not make the token read-only, then attributes can be added within your applications.

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom authorization token

   Because this login module relies on information in the sharedState added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule. For information on how to add your custom login module to the existing login configurations, see Custom login module development for a system login configuration

After completing these steps, you have implemented a custom AuthorizationToken.

Example: com.ibm.wsspi.security.token.AuthorizationToken implementation:

Use this file to see an example of a AuthorizationToken implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.AuthorizationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom AuthorizationToken, see "Implementing a custom AuthorizationToken" on page 1076.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
```

```
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomAuthorizationTokenImpl implements com.ibm.wsspi.security.
   token.AuthorizationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
 private static long expire_period_in_millis = 2*60*60*1000;
  // 2 hours in millis, by default

/**
 * Constructor used to create initial AuthorizationToken instance
 */

 public CustomAuthorizationTokenImpl (String principal)
 {
  // Sets the principal in the token
  addAttribute("principal", principal);
  // Sets the token version
  addAttribute("version", "1");
  // Sets the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a
 * propagation login.
 */
 public CustomAuthorizationTokenImpl (byte[] token_bytes)
 {
  try
  {
   hashtable = (java.util.Hashtable) com.ibm.wsspi.security.token.
           WSOpaqueTokenHelper.deserialize(token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
```

```
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long.
 * @return long
 */
 public long getExpiration()
 {
  // Gets the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // The expiration is the first element. There should be only one expiration.
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
  // You can choose whether your token gets propagated. In some cases,
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal that this Token belongs to. If this is an authorization token,
 * this principal string must match the authentication token principal string or the
 * message will be rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // this might be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon the information that provider
 * considers makes this a unique token. This will be used for caching purposes
 * and might be used in combination with other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
```

```
 *
 * @return String
 */
 public String getUniqueID()
 {
  // if you don't want to affect the cache lookup, just return NULL here.
  // return null;

  String cacheKeyForThisToken = "dynamic attributes";

  // if you do want to affect the cache lookup, return a string of
     // attributes that you want factored into the lookup.
  return cacheKeyForThisToken;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
    // Do this if the object is set to read-only during login commit,
    // because this makes sure that no new data gets set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = com.ibm.wsspi.security.token.WSOpaqueTokenHelper.
                 serialize(hashtable);

    // You can deserialize this in the downstream login module using
          // WSOpaqueTokenHelper.deserialize()
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token used to identify the byte[] in the protocol message.
 * @return String
 */
 public String getName()
 {
  return this.getClass().getName();
 }

/**
 * Gets the version of the token as an short. This also is used to identify the
 * byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();
```

```
  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only. The implementation
 * needs to ensure that any setter methods check that this flag has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is read-only
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name and value pair. Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();
```

```
   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomAuthorizationTokenImpl deep_clone =
   new com.ibm.websphere.security.token.CustomAuthorizationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

Example: custom AuthorizationToken login module:

This file shows how to determine if the login is an initial login or a propagation login

```
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.)
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
```

```
{
    // (For more information on what 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.
 Callback callbacks[] = new Callback[1];
 callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles exception
 }

 // Receives the ArrayList of TokenHolder objects (the serialized tokens)
 List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

 if (authzTokenList != null)
 {
  // Iterates through the list looking for your custom token
  for (int i=0; i
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

         // Looks for the name and version of your custom AuthorizationToken
         // implementation
   if (tokenHolder.getName().equals("com.ibm.websphere.security.token.
            CustomAuthorizationTokenImpl") &&
       tokenHolder.getVersion() == 1)
   {
           // Passes the bytes into your custom AuthorizationToken constructor
           // to deserialize
    customAuthzToken = new
     com.ibm.websphere.security.token.CustomAuthorizationTokenImpl(
                 tokenHolder.getBytes());

   }
  }
 }
 else
    // This is not a propagation login. Create a new instance of your
    // AuthorizationToken implementation
 {
      // Gets the prinicpal from the default AuthenticationToken. This must match
      // all tokens.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

      // Adds a new custom authorization token. This is an initial login. Pass the
      // principal into the constructor
```

```
  customAuthzToken = new com.ibm.websphere.security.token.
          CustomAuthorizationTokenImpl(principal);

  // Adds any initial attributes
  if (customAuthzToken != null)
  {
   customAuthzToken.addAttribute("key1", "value1");
   customAuthzToken.addAttribute("key1", "value2");
   customAuthzToken.addAttribute("key2", "value1");
   customAuthzToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case something
    // happens during the login.
}

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.)

 if (customAut  // (hzToken != null)
 {
  // sSets the customAuthzToken token into the Subject
  try
  {
   public final AuthorizationToken customAuthzTokenPriv = customAuthzToken;
         // 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.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
                 // Adds the custom authorization token if it is not null
                 //  and not already in the Subject
                               if ((customAuthzTokenPriv != null) &&
        (!subject.getPrivateCredentials().contains(customAuthzTokenPriv)))
      {
       subject.getPrivateCredentials().add(customAuthzTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
```

```
   }
  }
 }

 // 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;
}
```

Default SingleSignonToken

Do not use the default SingleSignonToken in service provider code. This default token is used by the WebSphere Application Server run-time code only. There are size limitations 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 SingleSignonToken. To implement a custom SingleSignonToken, see “Implementing a custom SingleSignonToken” on page 1086 for more information.

Changing the TokenFactory associated with the default SingleSignonToken

When default SingleSignonToken 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 > Global Security**.
2. Under Additional properties, click **Custom properties**.

The default TokenFactory specified for this property is called com.ibm.ws.security.ltpa.LTPAToken2Factory. This token factory creates an SSO token called LtpaToken2, which WebSphere Application Server uses for propagation. This TokenFactory uses the AES/CBC/PKCS5Padding cipher. If you change this TokenFactory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to version 5.1.1 that use the default TokenFactory. Only servers running WebSphere Application Server Version 5.1.1 or later with propagation enabled are aware of the LtpaToken2 cookie. However, this is not a problem if all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new TokenFactory.

If you need to perform your own signing and encryption of the default SingleSignonToken, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the LTPA keys passed into the initialize method of the TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default SingleSignonToken using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
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 *install directory*/classes directory so that the WebSphere class loader can load the classes.

Implementing a custom SingleSignonToken

This task explains how to create your own SingleSignonToken implementation, which is set in the login Subject and added to the HTTP response as an HTTP cookie. The cookie name is the concatenation of the SingleSignonToken.getName() application programming interface (API) and the SingleSignonToken.getVersion() API. There is no delimiter. When you add a SingleSignonToken to the Subject, it also gets propagated horizontally and downstream in case the Subject is used for other Web requests. You must deserialize your custom SingleSignonToken when you receive it from a propagation login. Consider writing your own implementation if you want to accomplish one of the following:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. It is recommended that you encrypt the information because it is out to the HTTP response and is available on the Internet. You must deserialize or decrypt the bytes at the target and add that information back into the Subject.
- Affect the overall uniqueness of the Subject using the getUniqueID() API

To implement a custom SingleSignonToken, you must complete the following steps:

1. Write a custom implementation of the SingleSignonToken interface.

   There are many different methods for implementing the SingleSignonToken interface. However, make sure that the methods required by the SingleSignonToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the *install_dir*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so that it has the necessary permissions that are needed by the server code.

   Tip: All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the SingleSignonToken, might extend the abstract class and then most of the work is completed.

   To see an implementation of the SingleSignonToken, see “Example: com.ibm.wsspi.security.token.SingleSignonToken implementation” on page 1087
2. Add and receive the custom SingleSignonToken during WebSphere Application Server logins. This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, in order to deserialize the information, you will need to plug in a custom login module, which is discussed in a subsequent step. After the object is instantiated in the login module, you can add it to the Subject during the commit() method.

   The code sample in “Example: custom SingleSignonToken login module” on page 1092 shows how to determine if the login is an initial login or a propagation login. The difference is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom SingleSignonToken implementation and set it into the Subject. Also, look for the HTTP cookie from the HTTP request if the HTTP request object is available in the callback. You can get your custom SingleSignonToken both from a horizontal propagation login and from the HTTP request. However, it is recommended that you make the token available in both places because then the information arrives at any front-end application server even if that server that does not support propagation.

   You can make your SingleSignonToken read-only in the commit phase of the login module. If you make the token read-only, additional attributes cannot be added within your applications.

   Restriction:

   - HTTP cookies have a size limitation so do not add too much data to this token.

- The WebSphere Application Server run time does not handle cookies that it does not generate, so this cookie is not used by the run time.
- The SingleSignonToken object, when in the Subject, does affect the cache lookup of the Subject if you return something in the getUniqueID() method.

3. Get the HTTP cookie from the HTTP request object during login or from an application. The sample code, found in “Example: HTTP cookie retrieval” on page 1094 shows how you can retrieve the HTTP cookie from the HTTP request, decode the cookie so that it is back to your original bytes, and create your custom SingleSignonToken object from the bytes.
4. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom propagation token Because this login module relies on information in the sharedState added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule.

   For information on adding your custom login module into the existing login configurations, see Custom login module development for a system login configuration

After completing these steps, you have implemented a custom SingleSignonToken.

Example: com.ibm.wsspi.security.token.SingleSignonToken implementation:

Use this file to see an example of a SingleSignon implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.SingleSignonToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom SingleSignonToken, see “Implementing a custom SingleSignonToken” on page 1086.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomSingleSignonTokenImpl implements com.ibm.wsspi.security.
   token.SingleSignonToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
   // 2 hours in millis, by default
 private static long expire_period_in_millis = 2*60*60*1000;

/**
 * Constructor used to create initial SingleSignonToken instance
 */

 public CustomSingleSignonTokenImpl (String principal)
 {
  // set the principal in the token
  addAttribute("principal", principal);
```

```
  // set the token version
  addAttribute("version", "1");
  // set the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a propagation login.
 */
 public CustomSingleSignonTokenImpl (byte[] token_bytes)
 {
  try
  {
   // you should implement a decryption algorithm to decrypt the cookie bytes
   hashtable = (java.util.Hashtable) some_decryption_algorithm (token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it's does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long.
 * @return long
 */
 public long getExpiration()
 {
  // get the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // expiration will always be the first element (should only be one)
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
```

```
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
  // You can choose whether your token gets propagated or not, in some cases
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal that this Token belongs to. If this is an authorization token,
 * this principal string must match the authentication token principal string or the
 * message will be rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // this could be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon information the provider
 * considers makes this a unique token.  This will be used for caching purposes
 * and may be used in combination with outher token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
  // this could be any combination of attributes
  return getPrincipal();
 }

/**
 * Gets the bytes to be sent across the wire.  The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
    // do this if the object is set read-only during login commit,
    // since this guarantees no new data gets set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = some_encryption_algorithm (hashtable);

    // you can deserialize the tokenBytes using a similiar decryption algorithm.
```

```
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token, used to identify the byte[] in the protocol message.
 * @return String
 */
 public String getName()
 {
  return "myCookieName";
 }

/**
 * Gets the version of the token as an short.  This is also used to identify the
 * byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only.  The implementation
 * needs to ensure any setter methods check that this has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is readonly
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
```

```
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name/value pair.  Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // get the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // copy the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // allocate a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // add the String to the current array list
   array.add(value);

   // add the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // return the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the List of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomSingleSignonImpl deep_clone =
   new com.ibm.websphere.security.token.CustomSingleSignonTokenImpl();

  java.util.Enumeration keys = getAttributeNames();
```

```
  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

Example: custom SingleSignonToken login module:

This file shows how to determine if the login is an initial login or a propagation login

```
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.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // handle exception
  }

  // Receives the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

  if (authzTokenList != null)
  {
   // iterate through the list looking for your custom token
   for (int i=0; i
   for (int i=0; i<authzTokenList.size(); i++)
   {
    TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);
```

```
   // Looks for the name and version of your custom SingleSignonToken
         // implementation
   if (tokenHolder.getName().equals("myCookieName")
             && tokenHolder.getVersion() == 1)
   {
    // Passes the bytes into your custom SingleSignonToken constructor
           // to deserialize
    customSSOToken = new
     com.ibm.websphere.security.token.CustomSingleSignonTokenImpl
                 (tokenHolder.getBytes());

   }
  }
 }
 else
         // This is not a propagation login. Create a new instance of your
         // SingleSignonToken implementation
 {
      // Gets the principal from the default SingleSignonToken. This principal
      //  must match all tokens.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

  // Adds a new custom single signon (SSO) token. This is an initial login.
      //  Pass the principal into the constructor
  customSSOToken = new com.ibm.websphere.security.token.
          CustomSingleSignonTokenImpl(principal);

  // add any initial attributes
  if (customSSOToken != null)
  {
   customSSOToken.addAttribute("key1", "value1");
   customSSOToken.addAttribute("key1", "value2");
   customSSOToken.addAttribute("key2", "value1");
   customSSOToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case something
    // happens during the login.
}

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 (customSSOToken != null)
 {
  // Sets the customSSOToken token into the Subject
  try
  {
   public final SingleSignonToken customSSOTokenPriv = customSSOToken;
         // 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.PrivilegedAction()
    {
     public Object run()
     {
      try
      {
       // Adds the custom SSO token if it is not null and
                 //  not already in the Subject
                                if ((customSSOTokenPriv != null) &&
          (!subject.getPrivateCredentials().
                           contains(customSSOTokenPriv)))
       {
        subject.getPrivateCredentials().
                        add(customSSOTokenPriv);
       }
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.SingleSignonToken customSSOToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
 java.util.Map _sharedState = null;
}
```

Example: HTTP cookie retrieval:

Use this file to see an example of how to retrieve a cookie from an HTTP request, decode the cookie so that it is back to your original bytes, and create your custom SingleSignonToken object from the bytes. This example shows how to complete these steps from a login module. However, you also can complete these steps using a servlet.

For information on how to implement a custom SingleSignonToken, see “Implementing a custom SingleSignonToken” on page 1086.

```
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.)
```

```
 _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.
 Callback callbacks[] = new Callback[2];
 callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");
 callbacks[1] = new WSServletRequestCallback("HttpServletRequest: ");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles the exception
 }

 // receive the ArrayList of TokenHolder objects (the serialized tokens)
 List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();
 javax.servlet.http.HttpServletRequest request =
        ((WSServletRequestCallback) callbacks[1]).getHttpServletRequest();

 if (request != null)
 {

  // Checks if the cookie is present
  javax.servlet.http.Cookie[] cookies = request.getCookies();
  String[] cookieStrings = getCookieValues (cookies, "myCookeName1");

  if (cookieStrings != null)
  {
   String cookieVal = null;
   for (int n=0;n<cookieStrings.length;n++)
   {
    cookieVal = cookieStrings[n];
    if (cookieVal.length()>0)
    {
              // Removes the cookie encoding from the cookie to get
              // your custom bytes
     byte[] cookieBytes =
      com.ibm.websphere.security.WSSecurityHelper.
                    convertCookieStringToBytes(cookieVal);
     customSSOToken =
      new com.ibm.websphere.security.token.
                    CustomSingleSignonTokenImpl(cookieBytes);

              // Now that you have your cookie from the request,
              // you can do something with it here, or add it
              // to the Subject in the commit() method for use later.
     if (debug || tc.isDebugEnabled())
```

```
     {
      Systen.out.println("*** GOT MY CUSTOM SSO TOKEN FROM
                    THE REQUEST ***");
     }
    }
   }
  }
 }

}

public boolean commit() throws LoginException
{
    // (For more information on what to during a commit, see
    // Custom login module development for a system login configuration.)

 if (customSSOToken != null)
 {
  // Sets the customSSOToken token into the Subject
  try
  {
   public final SingleSignonToken customSSOTokenPriv = customSSOToken;
         // 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.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
                // Add the custom SSO token if it is not null and not
                // already in the Subject
                              if ((customSSOTokenPriv != null) &&
        (!subject.getPrivateCredentials().
                       contains(customSSOTokenPriv)))
      {
       subject.getPrivateCredentials().add(customSSOTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }
}

// Private method to get the specific cookie from the request
```

```
private String[] getCookieValues (Cookie[] cookies, String hdrName)
{
 Vector retValues = new Vector();
 int numMatches=0;
 if (cookies != null)
 {
  for (int i = 0; i < cookies.length; ++i)
  {
   if (hdrName.equals(cookies[i].getName()))
   {
    retValues.add(cookies[i].getValue());
    numMatches++;
    System.out.println(cookies[i].getValue());
   }
  }
 }

 if (retValues.size()>0)
  return (String[]) retValues.toArray(new String[numMatches]);
 else
  return null;
}

// Defines your login module variables
com.ibm.wsspi.security.token.SingleSignonToken customSSOToken = null;
com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
java.util.Map _sharedState = null;
}
```

Default AuthenticationToken

Do not use the default AuthenticationToken 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 AuthenticationToken” on page 1098 for more information.

Changing the TokenFactory associated with the default AuthenticationToken

When WebSphere Application Server generates a default AuthenticationToken, 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 > Global Security**.
2. Under Additional properties, click **Custom properties**.

The default TokenFactory specified for this property is called com.ibm.ws.security.ltpa.LTPATokenFactory. The LTPATokenFactory uses the DESede/ECB/PKCS5Padding cipher. This token factory creates an interoperable Lightweight Third Party Authentication (LTPA) token. If you change this TokenFactory, 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 TokenFactory implementation. However, this is not a problem if all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new TokenFactory.

If you associate com.ibm.ws.security.ltpa.LTPAToken2Factory with the com.ibm.wsspi.security.token.authenticationTokenFactory property, the token is AES encrypted. However,

you need to weigh the performance against your security needs. By doing this, you might add additional attributes to the AuthenticationToken in the Subject during a login that are available downstream.

If you need to perform your own signing and encryption of the default AuthenticationToken, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the LTPA keys passed into the initialize method of the TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default AuthenticationToken using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
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 TokenFactory implementation.
4. Verify that your implementation classes are put into the *install directory*/classes directory so that the WebSphere class loader can load the classes.

Implementing a custom AuthenticationToken

This task explains how you might create your own AuthenticationToken implementation, which is set in the login Subject and propagated downstream. This implementation enables you to specify an authentication token that can be used by a custom login module or application. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.
- Affect the overall uniqueness of the Subject using the getUniqueID() application programming interface (API).

Important: Custom AuthenticationToken implementations are not used by the security run time in WebSphere Application Server to enforce authentication. WebSphere Application Security run time uses this token in the following situations only:

- Call the getBytes() method for serialization
- Call the getForwardable() method to determine whether to serialize the AuthenticationToken.
- Call the getUniqueId() method for uniqueness
- Call the getName() and the getVersion() methods for adding serialized bytes to the TokenHolder that is sent downstream

All of the other uses are custom implementations.

To implement a custom authentication token, you must complete the following steps:

1. Write a custom implementation of the AuthenticationToken interface. There are many different methods for implementing the AuthenticationToken interface. However, make sure that the methods required by the AuthenticationToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the *install_dir*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader

can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the `server.policy` file so that it has the necessary permissions that are needed by the server code.

Tip: All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the AuthenticationToken, might extend the abstract class and then most of the work is completed.

To see an implementation of AuthenticationToken, see “Example: com.ibm.wsspi.security.token.AuthorizationToken implementation” on page 1077

2. Add and receive the custom AuthenticationToken during WebSphere Application Server logins This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, in order to deserialize the information, you must plug in a custom login module. After the object is instantiated in the login module, you can add the object to the Subject during the commit() method.

   If you only want to add information to the Subject to get propagated, see “Propagating a custom Java serializable object” on page 1107. If you want to ensure that the information is propagated, if you want to do your own custom serialization, or if you want to specify the uniqueness for Subject caching purposes, then consider writing your own AuthenticationToken implementation.

   The code sample in “Example: custom AuthenticationToken login module” on page 1104 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom AuthenticationToken implementation and set it into the Subject. If the callback contains propagation data, look for your specific custom AuthenticationToken TokenHolder instance, convert the byte[] back into your custom AuthenticationToken object, and set it back into the Subject. The code sample shows both instances.

   You can make your AuthenticationToken read-only in the commit phase of the login module. If you do not make the token read-only, then attributes can be added within your applications.

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom authorization token

   Because this login module relies on information in the sharedState added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule. For information on how to add your custom login module to the existing login configurations, see Custom login module development for a system login configuration

After completing these steps, you have implemented a custom AuthenticationToken.

Example: com.ibm.wsspi.security.token.AuthenticationToken implementation:

Use this file to see an example of a AuthenticationToken implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.AuthenticationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom AuthenticationToken, see “Implementing a custom AuthenticationToken” on page 1098.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
```

```
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomAuthenticationTokenImpl implements com.ibm.wsspi.security.
   token.AuthenticationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
  // 2 hours in millis, by default
   private static long expire_period_in_millis = 2*60*60*1000;
 private String oidName = "your_oid_name";
  // This string can really be anything if you do not want to use an OID.

/**
 * Constructor used to create initial AuthenticationToken instance
 */
 public CustomAuthenticationTokenImpl (String principal)
 {
  // Sets the principal in the token
  addAttribute("principal", principal);
  // Sets the token version
  addAttribute("version", "1");
  // Sets the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis()
        + expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a
 * propagation login.
 */
 public CustomAuthenticationTokenImpl (byte[] token_bytes)
 {
  try
  {
       // The data in token_bytes should be signed and encrypted if the
       // hashtable is acting as an authentication token.
   hashtable = (java.util.Hashtable) custom_decryption_algorithm (token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // If you set the expiration to 0, the token does not expire
  if (expiration != 0)
  {
   // Returns a response that identifies whether this token is still valid
```

```
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long type.
 * @return long
 */
 public long getExpiration()
 {
  // Gets the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // The expiration is the first element and there should only be one expiration
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
     // You can choose whether your token gets propagated. In some cases
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal to which this token belongs. If this is an
 * authorization token, this principal string must match the
 * authentication token principal string or the message is rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // This value might be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon information the provider
 * considers makes this a unique token. This identifier is used for caching purposes
```

```
 * and can be used in combination with other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
     // If you do not want to affect the cache lookup, just return NULL here.
     return null;

  String cacheKeyForThisToken = "dynamic attributes";

     // If you do want to affect the cache lookup, return a string of
     // attributes that you want factored into the lookup.
  return cacheKeyForThisToken;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
          // Do this if the object is set read-only during login commit
          // because this ensures that new data is not set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = custom_encryption_algorithm (hashtable);

    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token, which is used to identify the byte[] in the
 * protocol message.
 * @return String
 */
 public String getName()
 {
  return oidName;
 }

/**
 * Gets the version of the token as an short type.  This also is used
 * to identify the byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");
```

```
  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only.  The implementation
 * needs to ensure that any set methods check that this state has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is read-only
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name/value pair.  Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);
```

```
   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.wsspi.security.token.AuthenticationToken deep_clone =
   new com.ibm.websphere.security.token.CustomAuthenticationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }

/**
 * This method returns true if this token is storing a user ID and password
 * instead of a token.
 * @return boolean
 */
 public boolean isBasicAuth()
 {
  return false;
 }
}
```

Example: custom AuthenticationToken login module:

This file shows how to determine if the login is an initial login or a propagation login.

```
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.)
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
   // (For 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.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles exception
  }

  // Receives the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

  if (authzTokenList != null)
  {
   // Iterates through the list looking for your custom token
   for (int i=0; i<authzTokenList.size(); i++)
   {
    TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

          // Looks for the name and version of your custom AuthenticationToken
          // implementation
    if (tokenHolder.getName().equals("your_oid_name") && tokenHolder.getVersion() == 1)
    {
            // Passes the bytes into your custom AuthenticationToken constructor
            // to deserialize
     customAuthzToken = new
      com.ibm.websphere.security.token.
                CustomAuthenticationTokenImpl(tokenHolder.getBytes());

    }
   }
  }
  else
          // This is not a propagation login. Create a new instance of your
          // AuthenticationToken implementation
  {
```

```
      //  Gets the principal from the default AuthenticationToken. This principal
      //  should match all default tokens.
      //  Note: WebSphere Application Server run time only enforces this for
      //  default tokens. Thus, you can choose
      //  to do this for custom tokens, but it is not required.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

      // Adds a new custom authentication token. This is an initial login. Pass
      // the principal into the constructor
  customAuthToken = new com.ibm.websphere.security.token.
         CustomAuthenticationTokenImpl(principal);

  // Adds any initial attributes
  if (customAuthToken != null)
  {
   customAuthToken.addAttribute("key1", "value1");
   customAuthToken.addAttribute("key1", "value2");
   customAuthToken.addAttribute("key2", "value1");
   customAuthToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case
    // something happens during the login.
}

public boolean commit() throws LoginException
{
  // (For more information on what do during commit, see
  // Custom login module development for a system login configuration.)

 if (customAuthToken != null)
 {
  // Sets the customAuthToken token into the Subject
  try
  {
   private final AuthenticationToken customAuthTokenPriv = customAuthToken;
         // 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.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
                 // Adds the custom Authentication token if it is not
                 // null and not already in the Subject
                              if ((customAuthTokenPriv != null) &&
        (!subject.getPrivateCredentials().
                       contains(customAuthTokenPriv)))
      {
       subject.getPrivateCredentials().add(customAuthTokenPriv);
      }
     }
```

```
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.AuthenticationToken customAuthToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
 java.util.Map _sharedState = null;
}
```

Propagating a custom Java serializable object

Prior to completing this task, verify that security propagation is enabled in the administrative console.

With security attribute propagation enabled, you can propagate data either horizontally with single signon (SSO) enabled or downstream using Common Secure Interoperability version 2 (CSIv2). When a login occurs, either through an application login configuration or a system login configuration, a custom login module can be plugged in to add Java serializable objects into the Subject during login. This document describes how to add an object into the Subject from a login module and describes other infrastructure considerations to make sure that the Java object gets propagated.

1. Add your custom Java object into the Subject from a custom login module. There is a two-phase process for each Java Authentication and Authorization Service (JAAS) login module. WebSphere Application Server completes the following processes for each login module present in the configuration:

   login() method
   In this step, the login configuration callbacks are analyzed, if necessary, and the new objects or credentials are created.

   commit() method
   In this step, the objects or credentials that are created during login are added into the Subject.

   After a custom Java object is added into the Subject, WebSphere Application Server serializes the object on the sending server, deserializes the object on the receiving server, and adds the object back into the Subject downstream. However, there are some requirements for this process to occur successfully. For more information on the JAAS programming model, see the JAAS information provided in “Security: Resources for learning” on page 879.

   Important: Whenever you plug a custom login module into the login infrastructure of WebSphere Application Server, make sure that the code is trusted. When you add the login module into the *install_root*/classes directory, the login module has Java 2 Security AllPermissions. It is recommended that you add your login module and other infrastructure classes into any private directory. However, you must modify the *install_root*/properties/server.policy file to make sure that your private directory, Java archive (JAR) file, or both have the permissions needed to execute the application programming interfaces (API) that are called from the login module. Because the login

module might be executed after the application code on the call stack, you might add doPrivileged code so that you do not need to add additional properties to your applications.

The following code sample shows how to add doPrivileged:

```
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 phase, see
  // Custom login module development for a system login configuration.)

     // Construct callback for the WSTokenHolderCallback so that you
     // can determine if
     // your custom object has propagated
    Callback callbacks[] = new Callback[1];
    callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

    try
    {
         _callbackHandler.handle(callbacks);
    }
    catch (Exception e)
    {
    throw new LoginException (e.getLocalizedMessage());
    }

    // Checks to see if any information is propagated into this login
    List authzTokenList = ((WSTokenHolderCallback) callbacks[1]).
            getTokenHolderList();

    if (authzTokenList != null)
    {
        for (int i = 0; i< authzTokenList.size(); i++)
             {
            TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

                  //  Look for your custom object. Make sure you use
                  //  "startsWith"because there is some data appended
                  //  to the end of the name indicating in which Subject
                  //  Set it belongs. Example from getName():
                  //  "com.acme.CustomObject (1)". The class name is
                  //  generated at the sending side by calling the
                  //  object.getClass().getName() method. If this object
                  //  is deserialized by WebSphere Application Server,
                  //  then return it and you do not need to add it here.
                  //  Otherwise, you can add it below.
                  //  Note: If your class appears in this list and does
                  //  not use custom serialization (for example, an
```

```
                //  implementation of the Token interface described in
                //  the Propagation Token Framework), then WebSphere
                //  Application  Server automatically deserializes the
                //  Java object for you. You might just return here if
                //  it is found in the list.

          if (tokenHolder.getName().startsWith("com.acme.CustomObject"))
         return true;
       }
    }
       //  If you get to this point, then your custom object has not propagated
        myCustomObject = new com.acme.CustomObject();
        myCustomObject.put("mykey", "mydata");
}

public boolean commit() throws LoginException
{
 // (For more information on what to do during the commit phase, see
 // Custom login module development for a system login configuration.)

 try
 {
      // Assigns a reference to a final variable so it can be used in
      // the doPrivileged block
  final com.acme.CustomObject myCustomObjectFinal = myCustomObject;
  // Prevents your applications from needing a JAAS getPrivateCredential
      // permission.
  java.security.AccessController.doPrivileged(new java.security.
          PrivilegedExceptionAction()
  {
   public Object run() throws java.lang.Exception
   {
           // Try not to add a null object to the Subject or an object
           // that already exists.
    if (myCustomObjectFinal != null && !subject.getPrivateCredentials().
               contains(myCustomObjectFinal))
    {
              // This call requires a special Java 2 Security permission,
              // see the JAAS Javadoc.
     subject.getPrivateCredentials().add(myCustomObjectFinal);
    }
    return null;
   }
  });
 }
 catch (java.security.PrivilegedActionException e)
 {
  // Wraps the exception in a WSLoginFailedException
  java.lang.Throwable myException = e.getException();
  throw new WSLoginFailedException (myException.getMessage(), myException);
 }
}

// Defines your login module variables
com.acme.CustomObject myCustomObject = null;
}
```

2. Verify that your custom Java class implements the `java.io.Serializable` interface. An object that is added to the Subject must be serializable if you want the object to propagate. For example, the object must implement the `java.io.Serializable` interface. If the object is not serializable, the request does not fail, but the object does not propagate. To make sure that an object added to the Subject is propagated, implement one of the token interfaces defined in the “Security attribute propagation” on page 1052 article or add attributes to one of the following existing default token implementations:

   AuthorizationToken
   Add attributes if they are user-specific. For more information, see “Default AuthorizationToken” on page 1073.

   PropagationToken
   Add attributes that are specific to an invocation. For more information, see “Default PropagationToken” on page 1058.

   If you are careful adding custom objects and follow all the steps to make sure that WebSphere Application Server can serialize and deserialize the object at each hop, then it is sufficient to use custom Java objects only.
3. Verify that your custom Java class exists on all of the systems that might receive the request. When you add a custom object into the Subject and expect WebSphere Application Server to propagate the object, make sure that the class definition for that custom object exists in the *install_root*`/classes` directory on all of the nodes where serialization or deserialization might occur. Also, verify that the Java class versions are the same.
4. Verify that your custom login module is configured in all of the login configurations used in your environment where you would need to add your custom object during a login. Any login configuration that interacts with WebSphere Application Server generates a Subject that might be propagated outbound for an EJB request. If you want WebSphere Application Server to propagate a custom object in all cases, make sure that the custom login module is added to every login configuration that is used in your environment. For more information, see Custom login module development for a system login configuration.
5. Verify that security attribute propagation is enabled on all of the downstream servers that receive the propagated information. When an EJB request is sent to a downstream server and security attribute propagation is disabled on that server, only the authentication token is sent for backwards compatibility. Therefore, you must review the configuration to verify that propagation is enabled in all of the cells that might receive requests. There are several places in the administrative console that you must check to make sure propagation is fully enabled. For more information, see “Enabling security attribute propagation” on page 1056.
6. Add any custom objects to the propagation exclude list that you do not want to propagate. You can configure a property to exclude the propagation of objects that match specific class names, package names, or both. For example, you can have a custom object that is related to a specific process. If the object is propagated, it does not contain valid information. You must tell WebSphere Application Server not to propagate this object. Complete the following steps to specify the object in the propagation exclude list, using the administrative console:
   a. Click **Security > Global Security**.
   b. Under Additional Properties, click **Custom Properties > New**.
   c. Add `com.ibm.ws.security.propagationExcludeList` in the **Name** field.
   d. Add the name of the custom object in the **Value** field. You can add a list of custom objects to the propagation exclude list separated by a colon. For example, you might enter `com.acme.CustomLocalObject:com.acme.private.*`. You can enter a class name such as `com.acme.CustomLocalObject` or a package name such as `com.acme.private.*`. In this example, WebSphere Application Server does not propagate any class that equals `com.acme.CustomLocalObject` or begins with `com.acme.private.`.

      Although you can add custom objects to the propagation exclude list, you must be aware of a side effect. WebSphere Application Server stores the opaque token, or the serialized Subject contents, in a local cache for the life of the single signon (SSO) token. The life of the SSO token, which has

a default of two hours, is configured in the SSO properties on the administrative console. The information that is added to the opaque token includes only the objects not in the exclude list. If your authentication cache does not match your SSO token timeout, you might get a Subject on the local server that is regenerated from the opaque token but does not contain the objects on the exclude list. The authentication cache, which has a default of ten minutes, is configured on the Global Security panel on the administrative console. It is recommended that you make your authentication cache timeout value equal to the SSO token timeout so that the Subject contents are consistent locally.

As a result of this task, custom Java serializable objects are propagated horizontally or downstream. For more information on the differences between horizontal and downstream propagation, see “Security attribute propagation” on page 1052.

Authorization in WebSphere Application Server

WebSphere Application Server supports authorization 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. The default JACC provider that is provided by WebSphere Application Server uses the Tivoli Access Manager as the authorization provider.

When security is enabled in the 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 setup to be is used by WebSphere Application Server, all of the Enterprise JavaBeans (EJB) and Web authorization decisions are then 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 EJB components are protected and authorized per the J2EE specification. The administrative components are internal to WebSphere Application Server, and are protected by the RoleBasedAuthorizer. The administrative components include the admininstrative console application, MBeans, and other components such as naming and security. For more information on administrative security, see Role-based authorization.

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, so there should be no differences in function. Choose a JACC provider only when you want to work with an external security provider such as the Tivoli Access Manager. In this instance, the security provider must support the JACC specification and be set up to work with the 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 providers

The Java Authorization Contract for Containers (JACC) is a new specification introduced in Java 2 Platform, Enterprise Edition (J2EE) 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. The default (out-of-box) solution is the WebSphere Application Server default J2EE role based authorization implementation, which does not implement the JACC Policy provider interface.

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. All are required by the JACC specification.

The JACC specification does not specify how to handle the authorization table (user or group to role) information between the container and the provider. It is the responsibility of the provider to provide some management facilities to handle this information. It does not require the container to provide the authorization table information in the binding file to the provider.

WebSphere Application Server provides two role configuration interfaces (RoleConfigurationFactory and RoleConfiguration) 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 used to support JACC” on page 1123 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 server. 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 network deployment (ND) package.

The JACC provider is not an out-of-box solution. You must configure WebSphere Application Server to use the JACC provider.

Authorization providers settings:

Use this page to enable a Java Authorization Contract for Containers (JACC) provider for authorization decisions.

To view this administrative console page, click **Security** > **Global security**. Under Authorization, click **Authorization providers**.

WebSphere Application Server provides a default authorization engine that performs all of the authorization decisions. In addition, WebSphere 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 WebSphere 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 be used with WebSphere Application Server, do not enable **External authorization using JACC**.

Default authorization:

This option should be used all the time unless you want an external security provider such as the Tivoli Access Manager to perform the authorization decision for J2EE applications based on the JACC specification.

Default: Enabled

External authorization using a JACC provider:

Enable this option only when you plan to use an external security provider such as the Tivoli Access Manager for performing authorization decisions for J2EE applications using the JACC specification.

To use an external provider, you must complete the following steps:

1. Configure your JACC provider.
2. Verify that the required provider implementation classes are in the class path for each WebSphere Application Server process.

   Attention: This step is not required when you use Tivoli Access Manager because the application server already contains the implementation classes.
3. Enable the **External authorization using a JACC provider** option
4. Enter the appropriate properties for the provider under the External JACC provider link, which is located under Related Items.

Default: Disabled

External JACC provider: Use this link to configure WebSphere 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 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.

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 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 JavaBean (EJB) container or Web container calls the security run time 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 implemented by the provider to make the access decision.

The following sections describe how the provider is called for both the EJB 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. It creates the EJBMethodPermission object using the bean name, method name, interface name and the method signature.
2. It creates the contextID and sets it on the thread by using the PolicyContext.setContextID(contextID) method.
3. It registers the required policy context handlers, including the Subject policy context handler.
4. It creates the ProtectionDomain object with principal in the Subject. If there is no principal, 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.

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) is accessed, the security runtime delegates the authorization decision to the JACC provider by using the following process:

1. A WebResourcePermission is created to see if the URI is unchecked. If the provider honors the Everyone subject it should also be checked here.
   a. The WebResourcePermission is constructed with urlPattern and the HTTP method accessed.
   b. A ProtectionDomain with a null principal name is created.
   c. The JACC provider's Policy.implies( ) method is called with the permission and the protection domain. If the URI access is unchecked (or given access to Everyone subject), the provider should permit access (return true) in the implies() method. Access is then granted without further checks.
2. If the access was not granted in Step 1, a WebUserDataPermission is created and used to see if the Uniform Resource Identifier (URI) is precluded or excluded or must be redirected using HTTPS protocol.
   a. The WebUserDataPermission is constructed with the urlPattern accessed, along with 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. ProtectionDomain with a null principal name is created.
   c. The JACC provider's Policy.implies( ) method is called with the permission and the protection domain. If the request is using the HTTPS protocol and the implies returns false, the HTTP 403 error is returned to imply excluded/precluded permission and 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, LTPAToken), 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 has been granted access privileges to the URI.
   a. As in Step 1, the WebResourcePermission is created. The ProtectionDomain 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's implies() method is called using the Permission object and the ProtectionDomain created above. If the user is granted permission to access the resource, the implies( ) method should return true. If the user is not granted access, the implies() method should return false.

Note: Even if the order listed above is changed later (for example, to improve performance) the end result should be 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 credential of type WSCredential . 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 then used to make the access decision.

If the provider adds 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 information to the Subject. For more information, see “Enabling security attribute propagation” on page 1056.

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/or 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 they will support the dynamic updates by configuring the **supports dynamic module updates** option in the JACC configuration model (see “Configuring a JACC provider” on page 1122 for more information). This option can be enabled or disabled using the administrative console or by scripting. It is expected that most providers will 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 1116.

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), they can implement the com.ibm.wsspi.security.authorization.InitializeJACCProvider interface. See “Interfaces used to support JACC” on page 1123 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 should be 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. Previous versions of the WebSphere Application Server do not support this feature. 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 6.0.x nodes only. This implies that a mixed node environment containing a set of 5.x nodes in a 6.0 cell is not supported.

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 of the arguments for each EJB method.

If the provider supports and requires this context handler, enable 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 are effective after the servers have been restarted . By default this is disabled. When using the Tivoli Access Manager as the JACC provider, this option should be **disabled**, since 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 then propagated to the JACC provider using the JACC specification APIs.

WebSphere Application Server makes the contextID unique by using the string `href:cellName/appName/moduleName` as the contextID format for the modules. The `href` part of the string indicates that a hierarchical name is passed as the contextID.

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 contextID for the modules in the application contain `base1` (not `cell1`) as the cell name since the application was initially installed in `base1`.

The `appName` part of the string in the contextID represents the application name containing the module. The `moduleName` refers to the name of the module.

As an example, the contextID for the module Increment.jar in an application named DefaultApplication that is installed in `cell1` is href:cell1/DefaultApplication/Increment.jar.

JACC policy propagation: When an application is installed or deployed in the WebSphere Application Server, the security policy information in the application is propagated to the provider when the configuration is saved. The contextID for the application is saved in its application.xml file, used for propagating the policy to the JACC provider, and also for access decisions for 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 has implemented the RoleConfiguration interface, the security policy information propagated to the policy provider also contains the authorization table information. See “Interfaces used to support JACC” on page 1123 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.

Once an application is installed, it can be updated without first 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.

The security information should not be propagated to the provider during application updates, you can set the JVM property com.ibm.websphere.security.jacc.propagateonappupdate to false in the deployment manager (in ND) or the unmanaged base application server. If this property is set to false, then 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 ear file is used to replace the existing one, the provider is then refreshed with the entire application security policy information.

In the network deployment (ND) environment, when an application is installed and saved, the security policy information in that application is updated in the provider from the deployment manager (dmgr or cell). The application is not propagated to its respective nodes until the synchronization command is issued and completed. Also, in the ND environment when an application is uninstalled and saved at the deployment manager, the policy for that application is removed from the JACC provider. However, unless the synchronization command is issued and completed from the deployment manager to the nodes hosting the application, the applications are still running in the respective nodes. In this instance, any access to this application should be denied since the JACC provider does not contain the required information to make the access decision for that application. Note that any updates to the application already installed as described above are also propagated to the provider from the deployment manager. The changes in the provider are not in sync with the applications in the nodes until the synchronization is completed.

JACC registration of the provider implementation classes: The JACC specification states that providers can plug in their provider using the system properties `javax.security.jacc.policy.provider` and `javax.security.jacc.PolicyConfigurationFactory.provider`.

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‘s PolicyConfigurationFactory implementation.

Although both system properties are supported in WebSphere Application Server, it is highly recommended that you use the configuration model provided. You can set these values using either the JACC configuration panel (see “Configuring a JACC provider” on page 1122 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.

This is especially true in the case of a network deployment (ND) environment where multiple application servers can exist in the configuration. If the system properties are used, you must ensure that each of the Java virtual machine (JVM) processes in the configuration should set these properties. If the configuration model is used, the information is propagated to all processes through the synchronization process of the application server.

Enabling an external JACC provider

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.

To enable an external JACC provider using the administrative console:

1. From the WebSphere Application Server administrative console, click **Security** > **Global security**.
2. Under Authorization, click **Authorization providers**.
3. Under Related items, click **External JACC provider**.
4. 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.
5. 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.
6. Select the **External authorization using a JACC provider** option and click **OK**. To access this option, click **Security > Global Security**. Under Authorization, click **Authorization providers**.
7. Complete the remaining steps to enable global security. If you are using the Tivoli Access Manager you must select LDAP as the user registry. This same LDAP server should be used by the Tivoli Access Manager. For more information on configuring LDAP registries, see “Configuring Lightweight Directory Access Protocol user registries” on page 961.
8. In a Network Deployment (ND) environment only, make sure that all of the changes are synchronized across all nodes.
9. In a multinode environment, start the deployment manager configuration by issuing the following commands:

   ```
   install_dir/profiles/profile_name/bin/stopManager.bat -username user_name -password password
   install_dir/profiles/profile_name/bin/startManager.bat
   ```
10. Restart all servers to make these changes effective.

External Java Authorization Contract for Containers provider settings:

Use this page to configure WebSphere 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 WebSphere 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, which implies the default authorization, these settings are not used.

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

1. Click **Security > Global security**.
2. Under Authorization, click **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 WebSphere Application Server. Using the Tivoli Access Manager properties link under Additional properties, 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 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 WebSphere 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.

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 WebSphere 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 WebSphere 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.

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 WebSphere 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 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.

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 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 required by the provider.

These properties are propagated to the provider during the start up 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.

Tivoli Access Manager implementation does not require you to enter any properties in this link.

Tivoli Access Manager properties:

Specifies properties 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.

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 on. In this case, the security policy of the previously installed applications would not exist in the JACC provider to make the access decisions. You can reinstall all of the applications once JACC is enabled. The wsadmin scripting tool can be used to propagate information to the JACC provider independent of the application install 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. The wsadmin tool can be used to invoke this method at the deployment manager level.

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 “Interfaces used to support JACC” on page 1123 for more information about these interfaces.

The appNames 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 application(s) to the Tivoli Access Manager JACC provider, please create or import the users and groups that are in the application(s) to Tivoli Access Manager.
- Depending on the application or the number of applications propagated you might have to increase the request time-out period either in the soap.client.props (if using SOAP) or the sas.client.props (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 “Configuring a JACC provider” 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
wsadmin -user serverID -password serverPWD
set secadm [lindex [$AdminControl queryNames type=SecurityAdmin,*] 0]

// 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 a JACC provider

The Java Authorization Contract for Containers (JACC) defines a contract between Java 2 Platform, Enterprise Edition (J2EE) containers and authorization providers. It enables any third party authorization providers to plug into a J2EE 1.4 application server such as the WebSphere Application Server to make the authorization decisions when a J2EE resource is accessed. The JACC provider can be implemented using the Tivoli Access Manager.

Read the following articles for more detailed information about JACC before you attempt to configure the WebSphere Application Server to use a JACC provider:

- “JACC support in WebSphere Application Server” on page 1113
- “JACC providers” on page 1111
- “Tivoli Access Manager integration as the JACC provider” on page 1127

1. Start the WebSphere Application Server administrative console by clicking `http://yourhost.domain:9060/ibm/console` after starting the 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 is typically the server user ID specified when you configured the user registry).
2. Click **Security** > **Global Security** from the left navigation menu.
3. Under Authorization, click **Authorization Providers**.
4. Under General Properties, click **External JACC provider**.
5. Under Additional Properties, click **Tivoli Access Manager properties**.
6. Enter the following information:

   Enable embedded Tivoli Access Manager
   Select this option to enable the 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 point 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

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 form `policy_server:port`. 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 form `auth_server:port:priority`. The authorization server communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7136.

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.

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 (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 has not been 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 security administrator ID (for example, `cn=wasdmin, o=organization, c=country`). The ID name must match the Server user ID on the LDAP User Registry panel in the administrative console. To access the LDAP User Registry panel, click **Security** > **Global Security**. Under User registries, click **LDAP**.

After you have configured a JACC provider, you must enable it in the WebSphere Application Server administrative console. See “Enabling an external JACC provider” on page 1118 for more information.

Interfaces used to support JACC

WebSphere Application Server provides interfaces similar to PolicyConfigurationFactory and PolicyConfiguration so that the information that is stored in the bindings file can be propagated to the provider during installation. The interfaces are called RoleConfigurationFactory and RoleConfiguration . The implementation of these interfaces is optional.

RoleConfiguration

The RoleConfiguration interface is used to propagate the authorization information to the provider. This interface is similar to the PolicyConfiguration interface 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 install. 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 doesn't 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 doesn't 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 doesn't 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 doesn't 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

The RoleConfigurationFactory interface is similar to the PolicyConfigurationFactory interface introduced by JACC, and is used to obtain RoleConfiguration objects based on the contextIDs.

```
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 doesn't 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 doesn't 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
 * can't be obtained.
 **/
public abstract com.ibm.ws.security.policy.RoleConfiguration
         getRoleConfiguration(String contextID, boolean remove)
      throws RoleConfigurationException
```

InitializeJACCProvider

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 their 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 it, 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()
```

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. It 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 APIs.

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 used to support JACC” on page 1123 for more information about these interfaces.

Tivoli Access Manager client configuration

The Tivoli Access Manager client can be configured using either the administrative console or wsadmin scripting. The administrative console panels for the Tivoli Access Manager client configuration are located under the Security center panel. The Tivoli client must be set up to use the Tivoli JACC provider.

For more information about how to configure the Tivoli Access Manager client, see “Tivoli Access Manager JACC provider configuration” on page 1131.

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.

In a Network Deployment (ND) environment, the user or group-to-role mapping is on the application level, not on the node level.

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.

Tivoli Access Manager security for WebSphere Application Server

WebSphere Application Server Version 6 provides embedded IBM Tivoli Access Manager client technology to secure your WebSphere Application Server managed resources.

The benefits of using Tivoli Access Manager described here are only applicable when Tivoli Access Manager client code is used with the Tivoli Access Manager server:

Note: Tivoli Access Manager code is not embedded but bundled in some versions of WebSphere Application 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

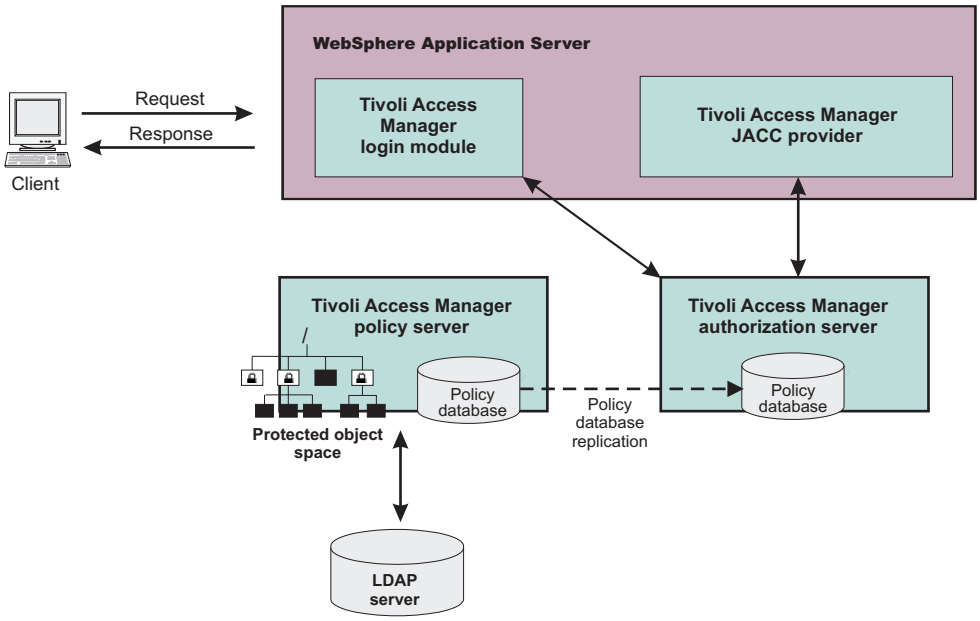
WebSphere Application Server Version 6.0.x supports the Java Authorization Contract for Containers (JACC) specification. JACC details the contract requirements for J2EE containers and authorization providers. With this contract, authorization providers can perform the access decisions for resources in J2EE Version 1.4 application servers such as WebSphere Application Server. The Tivoli Access Manager security utility that is embedded within WebSphere Application Server Version 6.0.x 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 (granted or denied) 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 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 information that is necessary to make deployment decisions, by reviewing the product documentation. Start with the following guides, available at http://publib.boulder.ibm.com/tividd/td/tdprodlist.html:

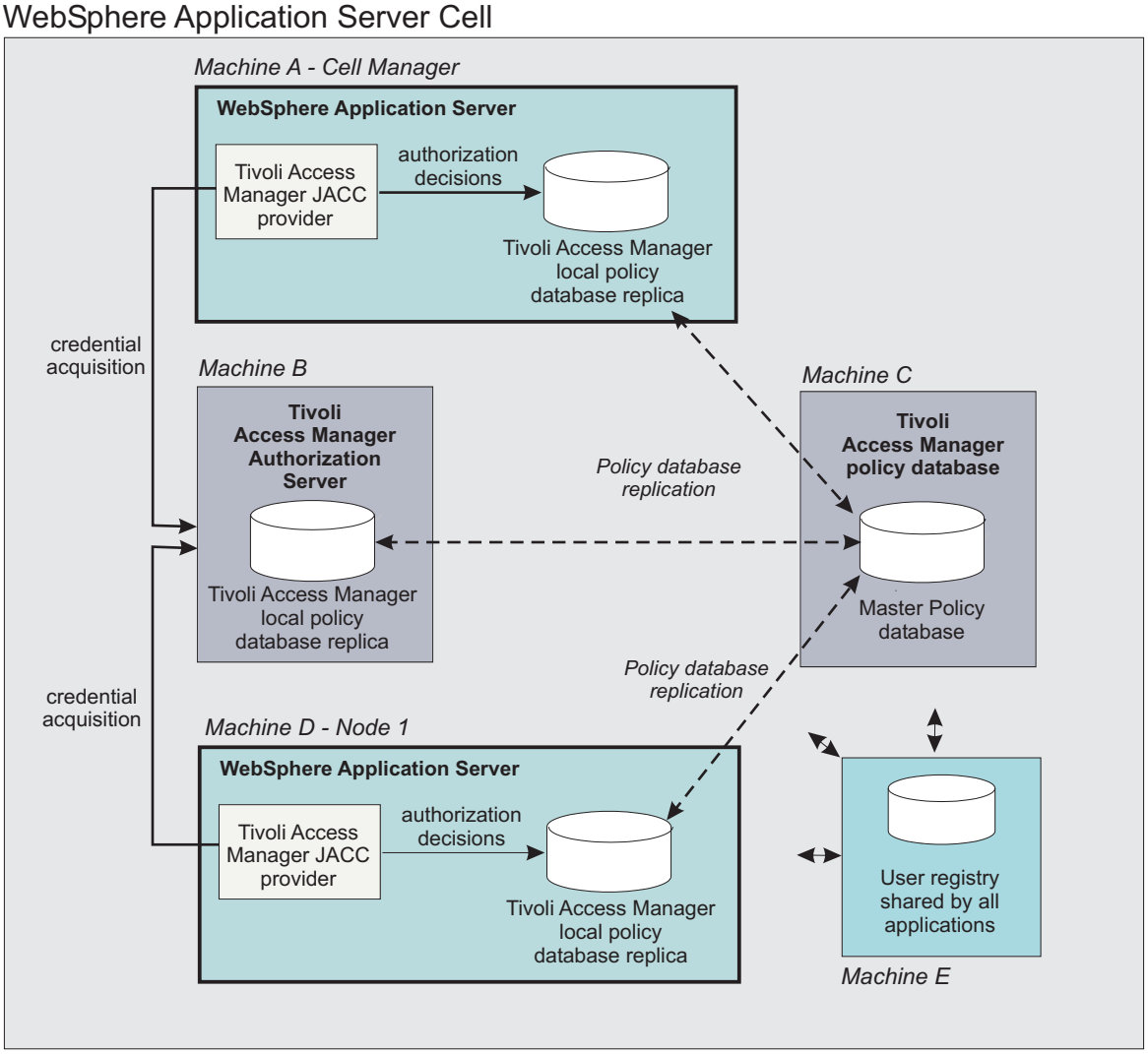
- *IBM Tivoli Access Manager Base Installation Guide*

  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.
- *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.

Tivoli Access Manager provides centralized administration of multiple servers.



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.

Note: It is possible to have separate WebSphere Application Server profiles on the same host configured against different Tivoli Access Manager servers. Such an architecture requires the profiles to be configured against separate Java Runtime Environments (JRE) and therefore multiple JREs need to be installed on the same host.

Creating the security administrative user

Enabling security requires the creation of a WebSphere Application Server administrative user. Use either the Tivoli Access Manager command-line pdadmin utility (available on the policy server host box) to create the Tivoli Access Manager administrative user for WebSphere Application Server. 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- “Tivoli Access Manager JACC provider configuration.”

Tivoli Access Manager JACC provider configuration

The Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider can be configured to deliver authentication and authorization protection for your applications or authentication only. Most deployments using the Tivoli Access Manager JACC provider will configure Tivoli Access Manager to 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 following property to the `amwas.amjacc.template.properties` file located on the directory `profiles/profile_name/cells/cell_name`.

```
com.tivoli.pd.as.amwas.DisableAddAuthorizationTableEntry=true
```

Once this property is set, perform the tasks for setting Tivoli Access Manager Security as documented.

You can configure the Tivoli Access Manager JACC provider using either the WebSphere Application Server administrative console or the **wsadmin** command line utility.

- For details on configuring the Tivoli Access Manager JACC provider using the administration console, refer to “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 1134

- 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”

Note:

Tivoli Access Manager JACC configuration files that are common across multiple WebSphere Application Server profiles are created by default under the `java/jre` directory. The user installing WebSphere Application Server will be given permissions to read and write to the files in this directory. On UNIX platforms, profiles created by users who are different to the user that installed the application will have read-only permissions for this directory. In addition, all users on the iSeries platform will have read-only access to this directory. This is not ideal as configuration of the Tivoli Access Manager JACC provider will fail in these situations.

To avoid this problem read and write permissions can be manually applied to the `java/jre` directory. For iSeries installations, however, the permissions for this directory cannot be changed. To avoid this situation the following property can be added to the `profiles/`*`profile_name`*`/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 sets the location of the Tivoli Access Manager JACC provider properties files that are common across profiles.

Note: The **wsadmin** command is available to reconfigure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) interface:

```
$AdminTask reconfigureTAM -interactive
```

This command effectively prompts you through the process of unconfiguring the interface and then reconfiguring it.

Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility

In a network deployment architecture, verify that all the managed servers, including node agents, are started. The following configuration is performed once on the deployment manager server. The configuration parameters are forwarded to managed servers, including node agents, when a synchronization is performed. The managed servers then require their own restart for the configuration changes to take effect.

You can use the wsadmin utility to configure Tivoli Access Manager security for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *`install_dir`*`/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 or enter an asterisk (*) to choose all nodes.

Option	Description
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 "Creating the security administrative user" on page 1131. 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.
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" on page 1137.

Configuring the JACC provider for Tivoli Access Manager using the administrative console

In a Network Deployment architecture, verify that all the managed servers, including node agents, are started. The following configuration is performed on the management server. When either **Apply** or **OK** is clicked, configuration information is checked for consistency, saved, and applied if successful. In Network Deployment environments, this configuration information is propagated to nodes when a synchronization is performed. Restart the nodes for the configuration changes to take effect.

To configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager using the administrative console:

1. Click **Security** > **Global security**.
2. Under Authorization, click **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:

Option	Description
Enable embedded Tivoli Access Manager	enable
Ignore errors during embedded Tivoli Access Manager disablement	This option is applicable only when reconfiguring an embedded Tivoli Access Manager client or when disabling an embedded Tivoli Access Manager client. When selected, errors are ignored during disablement of an embedded Tivoli Access Manager client.

Client 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, with each entry on a new line. If you specify a range of ports, separate the lower and higher values by a colon (:), as shown in the following example:
	
	`7999`
	`9990:9999`

Policy Server	Enter the name, the fully-qualified domain name, or the IP address of the Tivoli Access Manager policy server. Include the connection port. Use the form *policy_server* **:** *port*. The policy server communication port is set at the time of Tivoli Access Manager configuration – the default is 7135.

Option	Description

Authorization Servers	Enter the name, the fully-qualified domain name, or the 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. More than one authorization server can be specified by entering each server on a new line. 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.

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 identified previously.
User registry distinguished name suffix	Enter the distinguished name suffix for the user registry for Tivoli Access Manager and WebSphere Application Server to share. For example: o=*organization*,c=*country*
Security domain	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. 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 yet established at the time of Tivoli Access Manager configuration, leave the value as *Default*.
Administrator user distinguished name	Enter the full distinguished name of the WebSphere Application Server user ID, as created for Tivoli Access Manager in “Creating the security administrative user” on page 1131. For example, cn=wasadmin,o=*organization*,c=*country*. The name specified in this field must match the server user ID that is specified on the Lightweight Directory Access Protocol setting panel in the WebSphere Application Server administrative console. To access this panel, click **Security > Global security**. Under User registries, click **LDAP**.

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:

- Validate the configuration parameters.
- Configure 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.

Tivoli Access Manager JACC provider settings:

Use this page to configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager.

To view the JACC provider settings for Tivoli Access Manager, complete the following steps:

1. Click **Security** > **Global security**.
2. Under Authorization, click **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

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 reconfiguring 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.

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 use by the processes. If a range of ports is to be specified, 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
 9990:9999
```

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 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 was set at the time of Tivoli Access Manager configuration – the default is 7136. More than one authorization server can be specified by entering each server on a new line. 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.mycompany.com:7136:1
auth_server2.mycompany.com:7137:2
```

A priority (of 1) is still required when configuring against 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 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 WebSphere 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 WebSphere Application Server users and groups.

Specification of the Tivoli Access Manager domain is required as 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 full, distinguished name of the WebSphere Application Server security administrator ID. For example, cn=wasadmin,o=*organization*,c=*country*

Enabling the JACC provider for Tivoli Access Manager

Note: Do not perform this task if you are configuring the Java Authorization Contract for Container (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 > Global security**.
2. Under Authorization, click **Authorization providers**.
3. Select the **External authorization using a JACC provider** option.
4. Click **OK**. You are returned to the Authorization providers page.
5. Under Authorization, click **Authorization providers**.
6. Under Related Items, click **External JACC provider**. The JACC provider settings for Tivoli Access Manager are displayed.
7. 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 1118.
8. Under Additional properties, click **Tivoli Access Manager properties**.
9. 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 1136.
10. Click **OK**.
11. Save the settings by clicking **Save** at the top of the page.
12. Log out of the WebSphere Application Server administrative console.
13. Restart the 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.

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 from http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.
2. Reconfigure the Java Authorization Contract for Containers (JACC) provider using the `$AdminTask reconfigureTAM interactive` wsadmin command. Enter all new and existing options.

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.

Roles	Methods		
	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 invoked in the following table indicates that the principal or group can invoke any methods that are granted to that role.

Principal/Group	Roles		
	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 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.

Note: 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 Tivoli Access Manager JACC provider. This policy information contains permission-to-role mappings and role-to-principal and role-to-group mapping information.
2. The Tivoli Access Manager JACC provider 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 granted to this object is attached as an extended attribute.

Administering security users and roles with Tivoli Access Manager

User-to-role mapping and user-to-group mapping for the Tivoli Access Manager JACC provider are performed using the WebSphere Application Server administrative console. To manage user-to-role mappings and user-to-group mappings for applications:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Additional properties, click **Map security roles to Tivoli Access Manager users/groups**. The user and groups management screen is displayed.
3. Select the role which 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.

Configuring Tivoli Access Manager groups

The WebSphere Application Server administrative console can be used to specify security policies for applications that run in the WebSphere Application Server environment. The WebSphere Application Server administrative console can also specify security policies for other Web resources, based on the entities that are stored in the registry.

Tivoli Access Manager adds the accessGroup object class to the registry. Tivoli Access Manager administrators can use the pdadmin utility (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 > Global security**.
2. Under User registries, click **LDAP**.
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 then 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 then looks like the following example:

   ```
   groupOfNames:member;groupOfUniqueNames:uniqueMember;
   accessGroup:member
   ```

6. Stop and restart WebSphere Application Server.

Tivoli Access Manager JACC provider configuration properties

The Java property files are created in the WebSphere Application Server *install_dir*/profiles/*profiles_name*/etc/tam directory.

There are two properties files that may require configuration:

- **amwas.*node_server*.amjacc.properties** – contains properties used by the Tivoli Access Manager JACC provider.
- **amwas.*node_server*.pdjlog.properties** – contains logging properties created from the amwas.pdjlog.template.properties file for the specific node and server combination at the time of configuration.

Use **amwas.*node_server*.amjacc.properties** 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 file, amwas.*node_server*.amjacc.properties, located in the WebSphere Application Server *install_dir*/profiles/*profile_name*/etc/tam directory.

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 should not need to change this though the opportunity exists to implement your own cache if considered necessary.

Define static roles

```
com.tivoli.pd.as.cache.StaticRoleCache.Roles=Administrator,Operator,Monitor,Deployer
```

Defines the administration roles for WebSphere Application Server.

Note: Application performance can be enhanced by adding the static roles: **CosNamingRead**, **CosNamingWrite**, **CosNamingCreate**, **CosNamingDelete**. These roles allow 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 file, amwas.*node_server*.amjacc.properties, located in the WebSphere Application Server *install_dir*/profiles/*profile_name*/etc/tam directory.

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 should not need to change this though the opportunity exists to implement your own cache if considered 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 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 returned from a unique LDAP 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 file, `amwas.`*`node_server`*`.amjacc.properties`, located in the WebSphere Application Server *`install_dir`*`/profiles/`*`profile_name`*`/etc/tam` directory.

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:

The role-based policy framework parameters are located in the JACC configuration file and in the authorization configuration file. They 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 name of the configuration file generated from the authorization table is, `amwas.`*`node_server`*`.authztable.properties`. The name of the configuration file generated from the JACC provider is, `amwas.`*`node_server`*`.amjacc.properties`. Both files are stored on the WebSphere Application Server *`install_dir`*`/profiles/`*`profile_name`*`/etc/tam` directory. It is very unlikely that you will need to change these properties. They 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). It 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 specified by the `com.tivoli.pd.as.rbpf.AMAction` property. The permission set in `com.tivoli.pd.as.rbpf.AMAction` 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 (specified in the posroot property) to separate other products in the protected object space. Thus, embedded Tivoli Access Manager objects are found in the `/WebAppServer/deployedResources` directory and say AMWLS is in the `/WebAppServer/WLS` 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. If any of these properties 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 will cause access decisions to fail.

System-dependent configuration properties:

These properties are in the `amwas.`*`node_server`*`.amjacc.properties` file on the *`install_dir`*`/etc` directory. They should not be changed and are included here for reference only.

The supported arguments include :

com.tivoli.pd.as.rbpf.AmasSession.CfgURL=$WAS_HOME/profiles\profile_Name\etc\tam Files\\IBM\\WebSphere\\AppServer\\etc\\amwas.node_server.pdperm.properties
This entry is generated by the Java Authorization Contract for Containers (JACC) provider configuration. It specifies the location of the file containing information about the Tivoli Access Manager JACC provider. This entry should not be changed nor the properties in the file it points to.

com.tivoli.pd.as.rbpf.AmasSession.LoggingURL=file\:/C\:\\Program Files\\IBM\\WebSphere\\AppServer\\etc\\amwas.node_server.pdjlog.properties
This entry contains the location of the logging configuration file for the Tivoli Access Manager JACC provider. The file referenced is generated by the Tivoli Access Manager JACC provider configuration. This entry should not be changed.

Logging Tivoli Access Manager security

Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider messages are logged to the configured trace output location, and messages are written to standard out (`SystemOut.log`). When trace is enabled, all logging, both trace and messaging, is sent to `trace.log`.

The Tivoli Access Manager JACC provider uses the JLog logging framework as does the Tivoli Access Manager Java runtime environment. Tracing and messaging can be enabled selectively for specific Tivoli Access Manager JACC provider components.

Tracing and message logging for the Tivoli Access Manager JACC provider is configured in the properties file, `amwas.`*`node_server`*`.pdjlog.properties`, located on the `etc` directory. This file contains logging properties taken from the template file, `amwas.pdjlog.template.properties`, for the specific node and server combination at the time of Tivoli Access Manager JACC provider configuration.

The contents of this file lets the user control:

- Whether tracing is enabled or disabled for Tivoli Access Manager JACC provider components.

- Whether message logging is enabled or disabled for Tivoli Access Manager JACC provider components.

The amwas.*node_server*.pdjlog.properties file defines several *loggers*, each of which is associated with one Tivoli Access Manager JACC provider component. These loggers include:

AmasRBPFTraceLogger AmasRBPFMessageLogger	Used to log messages and trace for the role-based policy framework. This is an underlying framework used by embedded Tivoli Access Manager to make access decisions.
AmasCacheTraceLogger AmasCacheMessageLogger	Used to log messages and trace for the policy caches used by the role-based policy framework.
AMWASWebTraceLogger AMWASWebMessageLogger	Used to log messages and trace for the WebSphere Application Server authorization plug-in.
AMWASConfigTraceLogger AMWASConfigMessageLogger	Used to log messages and trace for the configuration actions for the Tivoli Access Manager JACC provider.
JACCTraceLogger JACCMessageLogger	Used to log messages and trace for Tivoli Access Manager JACC provider activity.

Note: Tracing can have a significant impact on system performance and should only be enabled 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's 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.

amwas.node_server.pdjlog.properties defines parent loggers called **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 Tivoli Access Manager JACC provider is enabled, the **isLogging** attribute is set to *true* for the **MessageLogger** and *false* for the **TraceLogger**. Message logging is therefore enabled for all components and tracing is disabled for all components by default.

To turn on tracing for a Tivoli Access Manager JACC provider component, two operations must occur:

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 Tivoli Access Manager JACC provider, the following line must be set to *true* in the amwas.*node_server*.pdjlog.properties:baseGroup.AMWASWebTraceLogger.isLogging=true
2. Enable tracing for the Tivoli Access Manager JACC provider 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. Under Logs and Trace tasks, click **Change Log Detail Levels**.
   g. Click **Components**. Tracing for all components can be enabled using **com.tivoli.pd.as.*** or tracing for separate components can be enabled using:
      - **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

h. Click **Apply**.

The trace specification should now indicate that tracing is enabled at the required level. Save the configuration, and restart the server for the changes to take effect.

Enabling embedded Tivoli Access Manager

Embedded Tivoli Access Manager is not enabled by default but needs to be configured 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 is the user registry containing the user and group information for both Tivoli Access Manager and WebSphere Application Server.
- A Tivoli Access Manager Version 5.1 domain exists and is configured to use the user registry. For details on the installation and configuration of Tivoli Access Manager refer to the: *Tivoli Access Manager Base installation Guide* and the *Tivoli Access Manager Base Administrator's Guide* available from http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.
- WebSphere Application Server Version 6.0.x is installed either in a single server model or as a network deployment.

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

1. Create the security administrative user. For more information, see "Creating the security administrative user" on page 1131.
2. Configure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider. For more information, see "Tivoli Access Manager JACC provider configuration" on page 1131.
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 "Configuring Lightweight Directory Access Protocol user registries" on page 961.
4. Enable the Tivoli Access Manager JACC provider. For more information, see "Enabling the JACC provider for Tivoli Access Manager" on page 1137.

Disabling embedded Tivoli Access Manager client

You can unconfigure Tivoli Access Manager Security in WebSphere Application Server using either the **wsadmin** command line utility or the WebSphere Application Server Administrative Console.

- For details on unconfiguring embedded Tivoli Access Manager client using the WebSphere Application Server Administration console, refer to "Disabling embedded Tivoli Access Manager client using the Administration Console."
- For details on unconfiguring embedded Tivoli Access Manager client using the **wsadmin** command line utility, refer to "Disabling embedded Tivoli Access Manager client using wsadmin" on page 1147.

Disabling embedded Tivoli Access Manager client using the Administration Console

In a network deployment architecture ensure all managed servers, including node agents, are started then perform the following process once on the deployment management server. Information from the unconfigure operation is forwarded to managed servers, including node agents, when the server is restarted. The managed servers then require their own restart for changes to take effect.

To unconfigure the Tivoli Access Manager JACC provider using the WebSphere Application Server administration console, complete the following steps:

1. Disable global security by clicking **Security** > **Global security** and deselect the **Enable global security** option.

2. Restart the server or, in a network deployment architecture, restart the deployment manager process.
3. Select **Security** > **Global security**.
4. Under Authorization, click **Authorization Providers**.
5. Under Related items, click **External JACC provider**.
6. Under Additional properties, click **Tivoli Access Manager Properties**. The configuration screen for the Tivoli Access Manager JACC provider is displayed.
7. Deselect 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.
8. Click **OK**.
9. **Optional:** If you want security enabled, without Tivoli Access Manager, re-enable global security.
10. **Optional:** In network deployment environments, synchronize all nodes.
11. Restart all WebSphere Application Server instances for the changes to take effect.

Disabling embedded Tivoli Access Manager client using wsadmin

In a network deployment architecture ensure all managed servers, including node agents, are started then perform the following process once on the deployment management server. Details of the unconfiguration are forwarded to managed servers, including node agents, when a synchronization is performed. The managed servers then require their own reboot for the configuration changes to take effect.

To unconfigure the Tivoli Access Manager JACC provider:

1. Disable global security by clicking **Security** > **Global security** and deselect the **Enable global security** option.
2. Restart the server or, in a network deployment architecture, restart the deployment manager process.
3. Start the **wsadmin** command line utility. The **wsadmin** command is found in install_dir\AppServer\bin.
4. 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 * to select all nodes.
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 name is usually, sec_master.
Tivoli Access Manager administrator's user password	Enter the password for the Tivoli Access Manager administrator.
Force	Enter *yes* if you want to ignore errors when unconfiguring the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider. 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.

5. When all information is entered, enter `F` to save the properties or `C` to cancel from the unconfiguration process and discard entered information.
6. **Optional:** If you want security enabled, not using Tivoli Access Manager, re-enable global security.
7. **Optional:** In network deployment environments, synchronize all nodes.

8. 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 Tivoli Access Manager JACC provider 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`. This effectively removes the Tivoli Access Manager JACC provider.

1. Back-up `security.xml` and `wsjaas.conf`.
2. Enter the following command as one continuous line:

```
WAS_HOME/java/jre/bin/java -classpath "WAS_HOME/lib/AMJACCProvider.jar:CLASSPATH"
com.tivoli.pd.as.jacc.cfg.CleanSecXML fully_qualified_path/security.xml
fully_qualified_path/wsjaas.conf
```

Updating console users and groups

Additions and changes to console users and groups are not automatically added to the Tivoli Access Manager object space once the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider is configured. Changes to console users and groups are saved in the `admin-authz.xml` file and this file will require migration before any changes take effect. The Tivoli Access Manger JACC provider includes the migration utility, migrateEAR, 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 Tivoli Access Manager JACC provider has been configured. The utility will not need to be run for changes and additions to console user and groups made prior to the Tivoli Access Manager JACC provider being configured as the changes (made to `admin-authz.xml`) are automatically migrated at configuration time. Furthermore, the migration tool does not need to be run before deploying standard J2EE applications, J2EE application policy deployment is also performed automatically.

To migrate `admin-authz.xml`:

1. Before executing the `migrateEAR` utility, setup the environment by running `setupCmdLine.bat` or `setupCmdLine.sh` located in the *installation*`/bin` directory.
2. Make sure that the *WAS_HOME* environment variable is set to the WebSphere Application Server installation directory.
3. Change to the directory where the `migrateEAR` utility is located: ${*WAS_HOME*}`/bin/`
4. Run the `migrateEAR` utility to migrate the data contained in `admin-authz.xml`. Use the parameter descriptions listed in “The Tivoli Access Manager migrateEAR utility.” For example:

   A status message is displayed when the migration completes. Output of the utility is logged to the file, `pdwas_migrate.log`, created on 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.
5. WebSphere Application Server does **not** require a restart for the changes to take effect.

The Tivoli Access Manager migrateEAR utility

Purpose

Migrates changes made to console users and groups in the `admin-authz.xml` file into the Tivoli Access Manager object space.

Syntax

```
migrateEAR
-j path
-c URI
-a admin_ID
-p admin_pwd
-w Websphere_admin_user
-d user_registry_domain_suffix
[-r root_objectspace_name]
[-t ssl_timeout]
```

Parameters

-a admin_ID
Specifies 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 URI
Specifies 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:

-d user_registry_domain_suffix
Specifies 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 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 path
Specifies the fully qualified path and file name of the Java 2 Platform Enterprise Edition application archive file. Optionally, this path can also be a directory of an expanded enterprise application. When WebSphere Application Server is installed in the default location, the paths to data files to migrate include:

-p admin_pwd
Specifies the password for the Tivoli Access Manager administrative user. The administrative user must have the privileges 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`.

This parameter is optional. When it is not specified, the user is prompted to supply the password for the administrative user name.

-r root_objectspace_name
Specifies 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. This parameter is optional.

The default value for the root object space is `WebAppServer`.

The Tivoli Access Manager root object space name is set by modifying the `amwas.amjacc.template.properties` prior to configuring the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider for the first time. This option should be used if the default object space value is not used in the configuration of the Tivoli Access Manager JACC provider.

The Tivoli Access Manager object space name should never be changed after the Tivoli Access Manager JACC provider has been configured.

-t ssl_timeout
Specifies 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 policy server before the default connection times out.

The default is 60 minutes. The minimum 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.

This parameter is optional. If you are not familiar with the administration of this value, you can safely use the default value.

-w WebSphere_admin_user
Specifies 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 1131. 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
Specifies the action group name. `WebAppServer` is the default name. 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 (enterprise archive files) to Tivoli Access Manager for WebSphere Application Server. The script calls the Java class: `com.tivoli.pdwas.migrate.Migrate`.

Before invoking the script you must run `setupCmdLine.bat` or `setupCmdLine.sh`. 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 containing the native language support libraries that are provided with the Tivoli Access Manager JACC provider. These libraries are located in a subdirectory under the Tivoli Access Manager JACC provider 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.

On Windows platforms, 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.

Return codes

The following exit status codes can be returned:

0 The command completed successfully.

1 The command failed.

Troubleshooting authorization providers

This article describes the issues you might encounter using a Java Authorization Contract for Containers (JACC) authorization provider. Tivoli Access Manager is bundled with WebSphere Application Server as an authorization provider. However, you also can plug in your own authorization provider.

Using Tivoli Access Manager as a Java Authorization Contract for Containers authorization provider

You might encounter the following issues when using Tivoli Access Manager as a JACC authorization provider:

- The configuration of JACC might fail.
- The server might fail to start after configuring JACC.
- The application might not deploy properly.
- The startServer command might fail after you have configured Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.
- An ″HPDIA0202w An unknown user name was presented to Access Manager″ error might occur.
- An ″HPDAC0778E The specified user's account is set to invalid″ error might occur.
- An WASX7017E: Exception received while running file ″InsuranceServicesSingle.jacl″ error might occur.

Using an external provider for Java Authorization Contract for Containers authorization

You might encounter the following issues when you use an external provider for JACC authorization:

- An ″HPDJA0506E Invalid argument: Null or zero-length user name field for the ACL entry″ error might occur.

The configuration of JACC might fail

If you are having problems configuring JACC, check the following:

- Ensure that the parameters are correct. For example, there should not be a number after TAM_Policy_server_hostname:7135, but there should be a number after TAM_Authorization_server_hostname:7136 (for example, TAM_Authorization_server_hostname:7136:1).
- If a message such as "server can't be contacted" appears, it is possible that the host names or port numbers of the Tivoli Access Manager servers are incorrect, or that the Tivoli Access Manager servers have not been started.
- Ensure that the password for sec_master is correct.
- Check the SystemOut.log and search for the string `AMAS` to see if any error messages are present.

The server might fail to start after configuring JACC

If the server does not start after JACC has been configured, check the following:

- Ensure that the WebSphere Application Server and Tivoli Access Manager use the same Lightweight Directory Access Protocol (LDAP) server.
- If the message "Policy Director Authentication failed″ appears, ensure that the:
  - WebSphere Application Server LDAP serverID is the same as the "Administrator user" in the Tivoli Access Manager JACC configuration panel.

- Tivoli Access Manager Administrator distinguished name (DN) is correct.
- Password of the Tivoli Access Manager administrator has not expired and is valid.
- Account is valid for the Tivoli Access Manager administrator.

- If a message such as “socket can‘t be opened for *xxxx*” (where *xxxx* is a number) appears, do the following:
  1. Go to $WAS_HOME/profiles/profile_name/etc/tam.
  2. Change *xxxx* to an available port number in amwas.commomconfig.properties, and amwas*cellName_dmgr.properties if dmgr failed to start. If Node failed to start, change *xxx* to an available port number in amwas*cellName_nodeName_.properties. If appSever failed to start, change *xxxx* in Amwas*cellname_nodeName_serverName.properties.

The application might not deploy properly

When you click **Save**, the policy and role information is propagated to the Tivoli Access Manager policy. It might take some time to finish. If the save fails, you must uninstall the application and then reinstall it.

To access an application after it is installed, you must wait 30 seconds (by default) to start the application after you save.

The startServer command might fail after you have configured Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.

If the cleanup for JACC unconfiguration or start server fails after JACC has been configured, do the following:

- Remove Tivoli Access Manager properties files from WebSphere Application Server. For each application server in a network deployment (ND) environment with N servers defined (for example, server1, server2), the following files must be removed:

  ```
  $WAS_INSTALL/java/jre/PdPerm.properties
  $WAS_INSTALL/java/jre/PdPerm.ks
  $WAS_INSTALL/profiles/profile_name/etc/tam/*
  ```

- Use a utility to clear the security configuration and return the system to the state it was in before Tivoli Access Manager JACC was configured. The utility removes all of the PDLoginModuleWrapper entries as well as the Tivoli Access Manager authorization table entry from the security.xml file, effectively removing the Tivoli Access Manager JACC provider. Backup security.xml before running this utility.

  Enter the following commands:

  ```
  $WAS_HOME/java/jre/bin/java -classpath
   "$WAS_HOME/lib/AMJACCProvider.jar:CLASSPATH"
  com.tivoli.pd.as.jacc.cfg.CleanSecXML fully_qualified_path/security.xml
  ```

An "HPDIA0202w An unknown user name was presented to Access Manager" error might occur

You might encounter the following error message if you are attempting to use an existing user in a Local Directory Access Protocol (LDAP) user registry with Tivoli Access Manager:

```
AWXJR0008E   Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E   A Tivoli Access Manager exception was caught. Details are:
"HPDIA0202W   An unknown user name was presented to Access Manager."
```

This problem might be caused by the hostname exceeding predefined limits with Tivoli Access Manager when it is configured against MS Active Directory. In WebSphere Version 6.0, the maximum length of the hostname can not exceed 46 characters.

Check that the hostname is not fully qualified. Configure the machine so that the hostname does not include the host domain.

To correct this error, complete the following steps:

1. On the command line, type the following information to get a Tivoli Access Manager command prompt:

   ```
   pdadmin -a administrator_name -p administrator_password
   ```

   The pdadmin *administrator_name* prompt is displayed. For example:

   ```
   pdadmin -a administrator1 -p password
   ```

2. At the pdadmin command prompt, import the user from the LDAP user registry to Tivoli Access Manager by typing the following information:

   ```
   user import user_name cn=user_name,o=organization_name,c=country
   ```

   For example:

   ```
   user import jstar cn=jstar,o=ibm,c=us
   ```

After importing the user to Tivoli Access Manager, you must use the `user modify` command to set the user account to valid. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

For information on how to import a group from LDAP to Tivoli Access Manager, see the Tivoli Access Manager documentation.

An "HPDAC0778E The specified user's account is set to invalid" error might occur

You might encounter the following error message after you import a user to Tivoli Access Manager and restart the client:

```
AWXJR0008E   Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E   A Tivoli Access Manager exception was caught.
Details are: "HPDAC0778E   The specified user's account is set to invalid."
```

To correct this error, use the `user modify` command to set the user account to valid. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

An "HPDJA0506E Invalid argument: Null or zero-length user name field for the ACL entry" error might occur

You might encounter an error similar to the following message when you propagate the security policy information from the application to the provider using the wsadmin command propagatePolicyToJACCProvider:

```
AWXJR0035E   An error occurred while  attempting to add member, cn=agent3,o=ibm,c=us, to role AgentRole
HPDJA0506E   Invalid argument: Null or zero-length user name field for the ACL entry
```

To correct this error, create or import the user, which is mapped to the security role to the Tivoli Access Manager. For more information on propagating the security policy information, see the documentation for your authorization provider.

An WASX7017E: Exception received while running file "InsuranceServicesSingle.jacl" error might occur

After the JACC provider and Tivoli Access Manager are enabled, when attempting to install the application (which is configured with security roles using the wsadmin command), the following error might occur:

```
WASX7017E: Exception received while running file "InsuranceServicesSingle.jacl"; exception information:
 com.ibm.ws.scripting.ScriptingException: WASX7111E: Cannot find a match for supplied option:
"[RuleManager, , , cn=mgr3,o=ibm,c=us|cn=agent3,o=ibm,c=us, cn=ManagerGro
up,o=ibm,c=us|cn=AgentGroup,o=ibm,c=us]" for task "MapRolesToUsers
```

The $AdminApp task option MapRolesToUsers becomes invalid when Tivoli Access Manager is used as the authorization server. To correct the error, change MapRolesToUsers to TAMMapRolesToUsers.

Authentication protocol for EJB security

In WebSphere Application Server Version 6, two authentication protocols are available to choose from: z/OS Secure Authentication Service (z/SAS) and Common Secure Interoperability Version 2 (CSIv2). 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 z/SAS and it is considered the strategic protocol.

Invoking EJB methods in a secure WebSphere Application Server environment requires an authentication protocol to determine the level of security and the type of authentication, which 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 (determined by the object Interoperable Object Reference (IOR)) with the client authentication requirements (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--SSL or TCP/IP?
- If Secure Sockets Layer (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?

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.

You can configure both protocols (z/SAS and CSIv2) to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for z/SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and z/SAS. If the client supports both and the server supports both, CSIv2 is used. However, if the server supports z/SAS (for example, it is a previous WebSphere Application Server release) and the client supports both, the client chooses z/SAS for this request because the z/SAS protocol is what both have in common.

CSIv2 is considered enabled on the client with the existence of the com.ibm.CORBA.ConfigURL java property. If the property is not specified or the specified property does not exist, CSIv2 is not enabled.

Common Secure Interoperability Specification, Version 2

The Common Secure Interoperability Specification, Version 2 (CSIv2) defines the Security Attribute Service (SAS) that enables interoperable authentication and delegation. The CSIv2 and z/SAS protocols are entirely different. The CSIv2 SAS is a subcomponent of CSIv2 that supports SSL and interoperability.

Security Attribute Service

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 used by WebSphere Application Server are add-on Interoperable Inter-ORB Protocol (IIOP) services. IIOP is a request-and-reply communications protocol 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 located on the server. As mentioned previously, this is where the authentication policy is established 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 (authenticated by the server), an authentication mechanism-specific token (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 reauthenticate or revalidate 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. The fact that a service context contains a unique identity is another way for WebSphere Application Server to support both SAS or z/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.

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 or z/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, 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 applied to the method and the roles given to the user. If the request is rejected by the security server, a reply is sent back to the client without ever invoking the business method.

If a service context is not found by the CSIv2 server request interceptor, the interceptor then looks at the transport connection to see if a client certificate chain was sent. This is done when SSL client authentication is configured between the client and server.

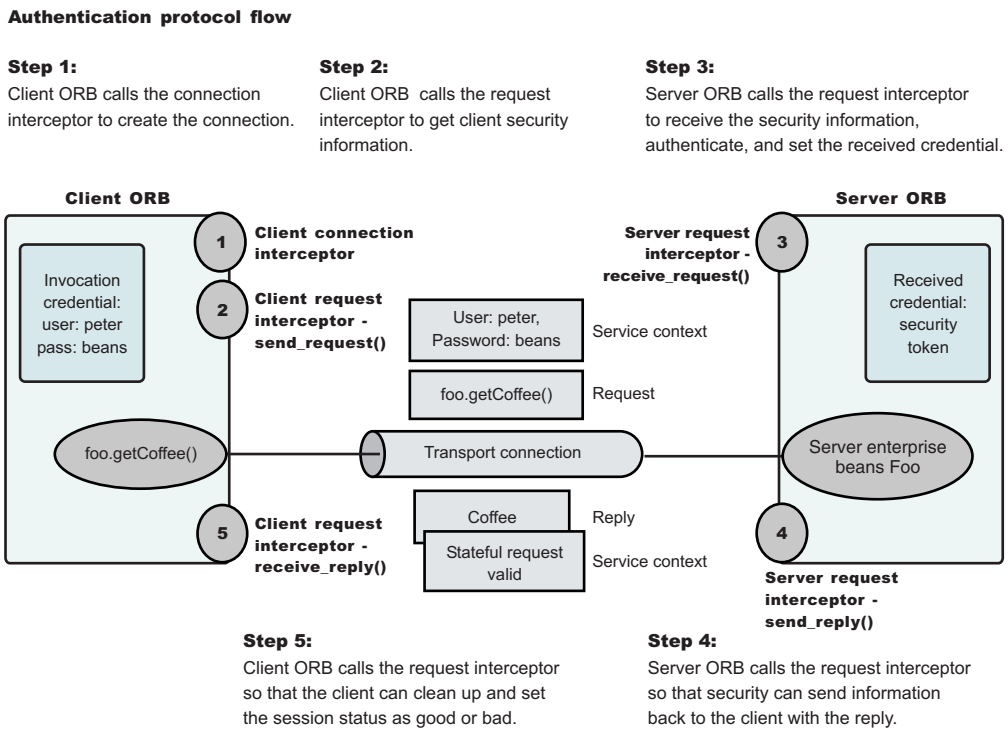
If the user registry is Lightweight Directory Access Protocol (LDAP), the search filters defined in the LDAP registry configuration determine how the certificate maps to an entry in the registry. If the user registry is local OS, the certificate is mapped to a System Authorization Facility (SAF) user ID. You then can map the user ID, using the issuers name or the subjects name, with the SAF certificate mapping facility.

If the certificate does not map, no credential is created and the request is rejected. When invalid security information is 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 or not. 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 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 the client side stateful context ID can be confirmed and reused.

The client and the server support both stateful and stateless sessions and this is not configurable.



. *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 only talk to servers that either require or support that feature. When a server requires a feature, it can only talk 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 requiring 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 run time 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.

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.
- SSL Client Certificate authentication.

  An additional way to authenticate a client to a server using SSL client authentication.
- 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 UserRegistry 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 (meaning this information cannot be located using the UserRegistry 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 sending server identity is sent using an SSL client certificate. If SSL is not used, the server identity is not sent.

Both 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, or it might require a real authenticated call. If the credentials of the sending server are authenticated and on the trusted principal list, then the server proceeds to evaluate the identity token.

The target server validates the authority of the sending server to assert an identity by the client certificate. The client certificate is mapped to a Service Access Facility (SAF) user ID. The user ID must have update authority for the `CBIND.servername` profile. If a client certificate is not sent, the CBIND check is performed against the default user ID.

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.

ITTDistinguishedName identity tokens and ITTCertChain identity tokens are mapped in the same way. Both types of identity tokens use a certificate that is mapped to a SAF user ID using the RACDCERT or equivalent mapping functions. The mapping can be based on the Subject name or the Issuers name.

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.

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 (GSSUP) as the authentication mechanism to establish client identity.

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
SWAM: No OID because it is not forwardable

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.CSI.performClientAuthenticationtype=SAFUSERIDPASSWORD

Basic authentication is currently the only valid value. You can configure the server through the administrative console.

While this property tells you which authentication mechanism to use, you also need to specify whether you want to perform authentication over the message layer, that is get a BasicAuth or a token-based credential. To complete this task, specify the com.ibm.CSI.performClientAuthenticationRequired (`True` or `False`) and com.ibm.CSI.performClientAuthenticationSupported (`True` or `False`) properties. Indicating that

client authentication is required implies that it must be done for every request. Indicating that the authentication mechanism is supported implies that it might be done, but is not required. For some servers, this option is appropriate if no resources are protected. In most cases, it is a best practice to indicate that this mechanism is supported so that client authentication is performed if both the client and server support it. Client authentication is not performed when communicating with certain servers that do not want security, yet the method requests still succeed.

Secure Sockets Layer client certificate authentication:

An additional way to authenticate a client to a server is using Secure Sockets Layer (SSL) client authentication.

Using SSL client authentication is another way of authenticating a client to a server. This form of authentication does not occur at the message layer using a user ID and password or tokens. This authentication occurs during the connection handshake using SSL certificates.

When the client is configured with a personal certificate in the key ring file, which indicates that SSL client authentication is required and the server supports SSL client authentication, the following actions occur to establish the identity on the client side.

- When a method request is invoked in the client code to a remote enterprise bean, the Object Request Broker (ORB) invokes the client connection interceptor to establish a connection with the server.

  Because the configuration specifies SSL and SSL client authentication, the connection type is SSL and the SSL handshake sends the client certificate to the server to validate. If the client certificate does not validate, the connection is not established and an exception is sent back to the client code where the method is invoked, which indicates the failure. If the client certificate is validated, then a connection opens between the client and the server.
- After the server receives the request, the server-side request interceptor checks for a security context.

  Because the server does not find a service context, it checks the server socket for a client certificate chain that contains the client identity. In this case, the server finds the certificate chain from the client. The identity in the certificate chain is valid because the connection is made. To create a credential, map the identity from the certificate to the user registry. This action is done differently based on the type of authentication mechanism. Mapping a certificate to a credential is done differently based on the user registry type.

One benefit of SSL client certificate authentication is that it optimizes authentication performance, because an SSL connection is typically created anyway. The extra overhead of sending the client certificate is minimal. While the client-side request interceptor performs no activity, the server-side request interceptor maps the certificate to a credential.

One disadvantage to this type of authentication is the complexity of setting up a key ring file on each client system.

To enable SSL client certificate authentication on the client side, you must set the properties. This action is completed using the following two properties:
- com.ibm.CSI.performTransportAssocSSLTLSRequired (true or false)
- com.ibm.CSI.performTransportAssocSSLTLSSupported (true or false)

Indicating that SSL is required implies that every request must generate an SSL connection key. If a server does not support SSL, then the request fails. After you enable SSL by either supporting it or requiring it, you can enable some of the SSL features.

To enable SSL client authentication, you can specify the following two properties:
- com.ibm.CSI.performTLClientAuthenticationRequired (true or false)
- com.ibm.CSI.performTLClientAuthenticationSupported (true or false)

The TL means *transport layer*. If you indicate that SSL client authentication is required, then you only limit the ability to communicate with servers that support SSL client authentication. For a server to support SSL client authentication, that server must have similarly configured properties through the administrative console, and have an SSL listener port that is open to handle mutual authentication handshakes. Configuration of server properties are done through the administrative console.

Supported authentication protocols: Two authentication protocols are supported by WebSphere Application Server. Secure Authentication Service (SAS) (or z/OS Secure Authentication Service (z/SAS) on the z/OS platform) is the authentication protocol used by all releases prior to WebSphere Application Server Version 5. Common Secure Interoperability Version 2 (CSIv2), which is considered the strategic protocol, is implemented in WebSphere Application Server, Version 5 and later.

In future releases, IBM will no longer ship or support the z/OS Secure Authentication Service (z/SAS) IIOP security protocol. It is suggested that you use the Common Secure Interoperability version 2 (CSIv2) protocols.

You can configure both protocols to work simultaneously. If a server supports both protocols, it exports an interoperable object reference (IOR) that contains tagged components describing the configuration for SAS or z/SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS or z/SAS. If the client and the server support both protocols, CSIv2 is used. However, if the server supports SAS or z/SAS (for example, it is a previous WebSphere Application Server release) and the client supports both protocols, the client chooses SAS or z/SAS for this request.

CSIv2 is considered enabled on the client with the existence of the com.ibm.CORBA.ConfigURL java property. If the property is not specified or the property does not exist, CSIv2 is not enabled.

Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols

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.

   A CSIv2 Java client utilizes a configuration file that is specified by the com.ibm.CORBA.ConfigURL Java property to configure outbound security.

   The upstream EJB application server configures inbound security to handle the right 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, then the outbound security might require a special configuration.
2. Specify the type of authentication.

   By default, the server supports authentication with a user ID and password.

   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, then consider how to configure CSIv2 to its best advantage.
3. Configure clients and servers.

   Configuring a pure Java client is done through a properties file that is specified by the com.ibm.CORBA.ConfigURL Java property.

   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.

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. The syntax of this property accepts any valid Web address.

```
-Dcom.ibm.CORBA.ConfigURL=file:/WebSphere/V5R0M0/AppServer/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 problems exist with the client properties file or there is no match with the server security, the Java client examines the server securities for non-Common Secure Interoperability Version 2 (CSIv2) security mechanisms that might be available. If no match is found with the old, non-CSIv2 securities either, the Java client attempts a nonsecure connection.

Use the following property to configure the z/SAS and CSIv2 authentication protocols:
- "CSIv2 authentication protocol client settings"

CSIv2 authentication protocol client settings: In addition to the properties that are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2), this page documents the properties that are valid for the CSIv2 protocol only.

com.ibm.CSI.performClientAuthenticationSupported:

Use to determine if message layer client authentication is supported.

When supported, message layer client authentication is performed when communicating with any server that supports or requires the authentication. Message layer client authentication involves transmitting either a user ID and password or a token from an already authenticated credential. If the authenticationTarget property is BasicAuth, the user ID and password are transmitted to the target server. If the authenticationTarget password is a token-based mechanism such as Lightweight Third Party Authentication (LTPA), then the credential token is transmitted to the server after authenticating the user ID and password directly to the security server.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performClientAuthenticationRequired:

Use to determine if message layer client authentication is required.

When required, message layer client authentication must occur when communicating with any server. If transport layer client authentication is also enabled, both authentications are performed, but message layer client authentication takes precedence at the server.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performTransportAssocSSLTLSSupported:

Use to determine if Secure Sockets Layer (SSL) is supported.

When SSL is supported, this client causes either SSL or TCP/IP to communicate with a server. If SSL is not supported, then the client must communicate over TCP/IP to the server. Supporting SSL is recommended so that any sensitive information is encrypted and digitally signed. When the associated com.ibm.CSI.performTransportAssocSSLTLSRequired property is enabled (set to `true`), this property is ignored. In this case, SSL is always required.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performTransportAssocSSLTLSRequired:

Use to determine if SSL is required.

When SSL is required, this client must use SSL to communicate to a server. If SSL is not supported by a server, this client does not attempt a connection to that server. When this property is enabled, the associated com.ibm.CSI.performTransportAssocSSLTLSSupported property is ignored.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performTLClientAuthenticationSupported:

Use to determine if transport layer client authentication is supported.

When performing client authentication using SSL, the client key file must have a personal certificate configured. Without a personal certificate, the client cannot authenticate to the server over SSL. If the personal certificate is a self-signed certificate, the server must contain the public key of the client in the server trust file. If the personal certificate is granted from a certificate authority (CA), the server must contain the root public key of the CA in the server trust file. This property is only valid when SSL is supported or required. If the associated com.ibm.CSI.performTLClientAuthenticationRequired property is enabled, this property is ignored.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performTLClientAuthenticationRequired:

Use to determine if transport layer client authentication is required.

If required, every secure socket that is opened between a client and server authenticates using SSL mutual authentication. When performing client authentication using SSL, the client key file must have a personal certificate configured. Without a personal certificate, the client cannot authenticate to the server over SSL.

If the personal certificate is a self-signed certificate, the server must contain the public key of the client in the server trust file. If the personal certificate is granted by a certificate authority (CA), the server must

contain the root public key of the CA in the server trust file. When this property is specified, the associated com.ibm.CSI.performTLClientAuthenticationSupported property is ignored.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performMessageConfidentialityRequired:

Use to determine if 128-bit ciphers must be used to make SSL connections.

If a target server does not support 128-bit ciphers, a connection to that server fails. This property is only valid when SSL is enabled. When this property is enabled, the associated com.ibm.CSI.performMessageConfidentialitySupported property is ignored.

Data type:	Boolean
Default:	True
Range:	True or False

com.ibm.CSI.performClientAuthenticationtype:

Use to define the type of client authentication.

The only value that is supported is `BasicAuth` .

Data type:	String constant
Default:	None
Range:	None

com.ibm.CSI.performSSL.Keyring: Used for providing the name of the Resource Access Control Facility (RACF) keyring used for SSL connections. Changes to this System Authorization Facility (SAF) keyring require changes to the `sas.client.props` file. For example, you might have to change the following properties:

- `com.ibm.ssl.keyStore=safkeyring:///WASKeyring`
- `com.ibm.ssl.trustStore=safkeyring:///WASKeyring`

Data type:	String
Default:	None
Range:	None

com.ibm.CORBA.loginUserid: Use to specify the user ID when a properties login is configured and message layer authentication occurs.

This property is only valid when `com.ibm.CORBA.loginSource=properties`. Also, set the com.ibm.CORBA.loginPassword property.

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 only valid when `com.ibm.CORBA.loginSource=properties`. Also, set the com.ibm.CORBA.loginUserid property.

Data type:	String
Range:	Any string that is appropriate for a password in the configured user registry of the server.

com.ibm.CSI.rmiOutboundPropagationEnabled: Enables the propagation of custom objects that are added to the Subject. On a pure client, add this property to the `sas.client.props` file. For more information, see Security Attribute Propagation.

z/OS Secure Authentication Service authentication settings:

Use this page to specify authentication settings for requests that are received and sent by a server that uses the z/OS authentication protocol. Use the z/OS Secure Authentication Service (z/SAS) protocol to communicate securely to enterprise beans with previous releases of WebSphere Application Server.

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

1. Click **Security** > **Global security**.
2. Under Authentication, click **Authentication Protocol** > **z/SAS authentication**.

You can also view this administrative console page by completing the following steps:

1. Click **Servers > Application Servers >** *server_name*.
2. Under Security, click **Server Security**.
3. Under Additional properties, click **z/SAS authentication**.

Note: z/SAS protocols are ignored unless the Active User Registry is Local OS.

Basic authentication:

Specifies that clients to this server can provide a System Authorization Facility (SAF) user ID and password over a Secure Sockets Layer (SSL) connection. This option requires a valid system SSL repertoire selection on the SSL settings option.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Client certificate:

Specifies that clients to this server can authenticate using SSL client certificates. The client certificates must be capable of mapping to a SAF user ID. You must connect the public certificate of the client certificate authority to the server key ring. The client certificate option requires a valid system SSL repertoire selection on the SSL settings option.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

User ID and password:

Specifies that clients can connect to this server with a SAF user ID and password without requiring a connection sent over an SSL session.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Identity assertion inbound:

Specifies that inbound requests using SAF user IDs that are forwarded by Application Server for z/OS can be accepted.

The immediate downstream server establishes its identity by sending a digital certificate. Identity assertion is available only if client certificates are supported. When you enable this setting, you must select an SSL setting.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Identity assertion outbound:

Specifies that outbound requests that originate from this server can forward authenticated client user IDs over an SSL connection to another Application Server for z/OS in which it has established trust.

This option requires a valid system SSL repertoire selection on the SSL settings option.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

Support unauthenticated clients:

Specifies that the server accepts Internet Inter-ORB Protocol (IIOP) requests without any authentication information.

If you enable this property, specify the Remote identity setting to associate a user ID with requests from a remote server.

Data type	Boolean
Default	Disabled
Range	Enabled or Disabled

SSL settings:

Specifies a predefined list of SSL settings for connections. Configure these settings on the SSL repertoire panel.

Data type	String
Default	None

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 > Global security**.
3. Under Authentication, click **Authentication Protocol > CSI inbound authentication**
4. Consider the following three 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 middle server identity is authorized on the target server with update authority on the CBIND class, profile`CB.<security prefix><cluster name>`. The middle server identity is sent using an SSL client certificate only. If SSL is not used, the CBIND check is performed against the configured default identity.
   - 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.
   - Secure Sockets Layer client certificate authentication (transport layer).

     This type of authentication typically occurs from pure clients using the certificate identity and from servers trusting the upstream server. Usually, when 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), not from the transport layer, through the client certificate authentication.

     A client has an SSL client certificate that is stored in the keystore file (or in the key ring file on the z/OS platform) 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 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 then 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, then 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, then a principal identity is sent. If authentication is done with a certificate, then the certificate chain is sent.

When you finish configuring this panel, you have configured most of the information that a client coalesces 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.

Attention: Although the session management list displays on the CSI Inbound Authentication panel, this option is not utilized by WebSphere Application Server for z/OS.

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. Refer to the article, 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, click **Security > Global security**. Under Authentication, click **Authentication protocols > CSIv2 inbound authentication**.

You can also view this administrative console page, by clicking **Servers > Application servers >***server_name*. Under Security, click **Server security**. 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 two 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.
- **Attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The attribute layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the attribute 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.

In the message layer, basic authentication (user ID and password) takes place. 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 select **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 LocalOS user registry, the certificate 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.

The following options are available for Client certificate authentication:

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.

Identity assertion:

Specifies that identity assertion is a way to assert identities from one server to another during a downstream Enterprise JavaBeans (EJB) invocation.

This server does not authenticate the asserted identity again because it trusts the upstream server. Identity assertion takes precedence over all other types of authentication.

Identity assertion is performed in the attribute layer and is only applicable on servers. The principal determined at the server is based on precedence rules. If identity assertion is performed, the identity is always derived from the attribute layer. If basic authentication is performed without identity assertion, the

identity is always derived from the message layer. Finally, if SSL client certificate authentication is performed without either basic authentication, or identity assertion, then the identity is derived from the transport layer.

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 one 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 sending server identity as a trusted identity through the Trusted Server IDs entry box. Enter a list of pipe-separated (|) principal names, for example, `serverid1|serverid2|serverid3`.

All identity token types map to the user ID field of the active user registry. For an ITTPrincipal identity token, this token maps one-to-one with the user ID fields. For an ITTDistinguishedName identity token, the value from the first equal sign is mapped to the user ID field. For an ITTCertChain identity token, the value from the first equal sign of the distinguished name is mapped to the user ID field.

When authenticating to an LDAP user registry, the LDAP filters determine how an identity of type ITTCertChain and ITTDistinguishedName get mapped to the registry. If the token type is ITTPrincipal, then the principal gets mapped to the UID field in the LDAP registry.

Data type: String

Trusted servers:

Specifies a semicolon-separated (;) or comma-separated (,) list of trusted server IDs, which are trusted to perform identity assertion to this server. For example, `serverid1;serverid2;serverid3` or `serverid1,serverid2,serverid3`.

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:

Specifies stateful sessions that 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 > Global security**. Under Authentication, click **JAAS configuration > System logins**.

Security attribute propagation:

Specifies whether to support security attribute propagation during login requests. When you select this option, WebSphere 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 > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**.

If you do not select this option, WebSphere Application Server does not accept any additional login information to propagate to downstream servers.

Additional Common Secure Interoperability inbound authentication settings:

Use this page to configure additional authentication settings for requests that are received by this server using the Object Management Group (OMG) Common Secure Interoperability authentication protocol.

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

1. Click **Security > Global security**.
2. Under Authentication, click **Authentication protocol** > **CSIv2 inbound authentication**.
3. Click **Additional settings** or **z/OS additional settings**.

You can also view this administrative console 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**. Click **Additional settings** or **z/OS additional settings**.

Client authentication type:

Specifies the type of client authentication supported for inbound requests.

Data type	String
Default	SAF user ID and password

SAF identity assertion:

Specifies that the server permits a trusted upstream server to assert client identities as System Authorization Facility (SAF) user names.

Data type	Boolean
Default	Disabled
Options	Enabled or Disabled

Distinguished name identity assertion:

Specifies that the server permits a trusted upstream server to assert client identities as distinguished names.

Note: This option is available for global security and not available for server-level security.

Data type	Boolean

Default	Disabled
Options	Enabled or Disabled

Certificate identity assertion:

Specifies that the server permits a trusted upstream server to assert client identities as X.509 certificates.

Note: This option is available for global security and not available for server-level security.

Data type	Boolean
Default	Disabled
Options	Enabled or Disabled

Configuring Common Secure Interoperability Version 2 outbound authentication:

Outbound authentication refers to the configuration that determines the type of authentication 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 run time 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.

1. 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 article for more information.
2. 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. Refer to the SSL Client Certificate Authentication article for more information.

   A server can send multiple layers simultaneously, therefore, 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. SSL client certificates are only used as the identity for invoking method requests, when that is the only layer provided. SSL client certificates are useful for trust purposes, even if the identity is not used for the request. If only the message layer and transport layer are provided, the message layer is used to establish the identity for authorization. If the identity assertion layer is provided (regardless of what is provided), then the identity from the identity token is always used by the authorization engine as the identity for that request.

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.

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 > Global security**.
2. Under Authentication, click **Authentication protocols > 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. Under Additional properties, 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.

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, 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.

The message layer for z/OS is empty.

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 client identity. If there are multiple identity types to assert, the identity is asserted in the following order: client certificate, distinguished name (DN), System Authorization Facility (SAF) user ID. The receiving server receives the identity in an identity token with an empty client authentication token. The Secure Sockets Layer (SSL) certificate of the server acts as the identity of the server to the receiving server.

Stateful sessions:

Specifies whether to reuse security information during authentication. This option is usually used to increase performance.

On z/OS systems, this option is ignored. The sending server prefers stateful sessions and uses them if the receiving server supports it.

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 clicking **Security > Global security**. Under Authentication, click **JAAS configuration > System login**.

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, click **Security > Global security**. Under Authentication, click **JAAS configuration > System logins > New**.

Security attribute propagation:

Enables WebSphere 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, click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**.

By default, the Security attribute propagation option is enabled and outbound login configuration is invoked. If you clear this option, WebSphere Application Server does not propagate any additional login information to downstream servers.

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.

Additional Common Secure Interoperability outbound authentication settings:

Use this page to configure additional authentication settings for requests that are received by this server using the Object Management Group (OMG) Common Secure Interoperability authentication protocol.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **CSIv2 outbound authentication**. Under Additional properties, click **Additional Settings** or **z/OS additional settings**.

You can also view this administrative console page by clicking **Servers > Application servers >** *server_name*. Under Security, click **Server security**. Under Additional properties, click **CSIv2 outbound authentication > Additional settings** or **CSIv2 outbound authentication > z/OS additional settings**.

Client authentication type:

Specifies the type of client authentication that is supported for outbound requests.

Data type	String
Default	SAF user ID and password

Configuring inbound transports:

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 z/OS Secure Authentication Service (z/SAS) have the ability to configure the transport.

CSIv2 and z/SAS support most of the same functions. CSIv2 has the advantage of interoperability with other WebSphere Application Server products and any other platforms that support the CSIv2 protocol.

Complete the following steps to configure the Inbound transport panels in the administrative console:

1. Click **Security > Global security**.
2. Under Authentication, click **Authentication Protocol > 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. These SSL settings are defined in the **Security** > **SSL** panel and define the SSL configuration including the key ring, security level, ciphers, and so on.

4. Consider fixing the listener ports that you configured.

   You complete this action in a different panel, but think about this action now. Most end points 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:
   - `ORB_SSL_LISTENER_ADDRESS` - SSL Port
   - `ORB_LISTENER_ADDRESS` - IIOP 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, which is generally not specified and selected dynamically during run time. 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.
5. Click **Security > Global security**. Under Authentication, click **Authentication protocol > z/SAS authentication** to select the SSL settings used for inbound requests from z/SAS clients.

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 > Global security**.
2. Under Authentication, click **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: 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.

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. You can select either **CSI** or **CSI and SAS** from the Global security panel, which is accessible from **Security > Global Security**.

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.

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 port. A zero port number indicates that a dynamic assignment is made at run time.

ORB_SSL_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. These settings are configured at the SSL Repertoire panel, which is accessible by clicking **Security > SSL**.

Data type: String
DefaultSSLSettings
Default: DefaultIIOPSSL
Range: Any SSL settings configured in the SSL Configuration Repertoire

Configuring outbound transports:

Outbound transports refers to the transport 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 keyrings and 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 > Global security**.
2. Under Authentication, click **Authentication protocols > 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 > Global security**. Under Authentication, click **Authentication Protocol > CSIv2 Outbound Transport**. By selecting the type of transport, you are choosing 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 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. Pick the SSL settings that correspond to an SSL transport. Click **Security > SSL**.

   Verify that the truststore keyring file in the selected SSL configuration contains the signers for any downstream servers. Also, verify that the downstream servers contain the server signer certificates when outbound client certificate authentication is used.

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 > Global security**.
2. Under Authentication, click **Authentication protocol > 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.

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, click **Security > SSL**.

Data type:	String
DefaultSSLSettings	
Default:	DefaultIIOPSSL
Range:	Any SSL settings that are configured in the SSL Configuration Repertoires panel

Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol provides transport layer security with authenticity, integrity, and confidentiality, for a secure connection between a client and server in WebSphere Application Server. The protocol runs above TCP/IP and below application protocols such as Hypertext Transfer Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), and Internet Inter-ORB Protocol (IIOP), and provides trust and privacy for the transport data.

Depending upon the SSL configurations of both the client and server, various levels of trust, data integrity, and privacy can be established. Understanding the basic operation of SSL is very important to proper configuration and to achieve the required protection level for both client and application data.

Some of the security features that are provided by SSL are data encryption to prevent the exposure of sensitive information while data flows. Data signing prevents unauthorized modification of data while data flows. Client and server authentication ensures that you talk to the appropriate person or machine. SSL can be effective in securing an enterprise environment.

SSL is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP transport, the Object Request Broker (ORB), and the secure LDAP client.

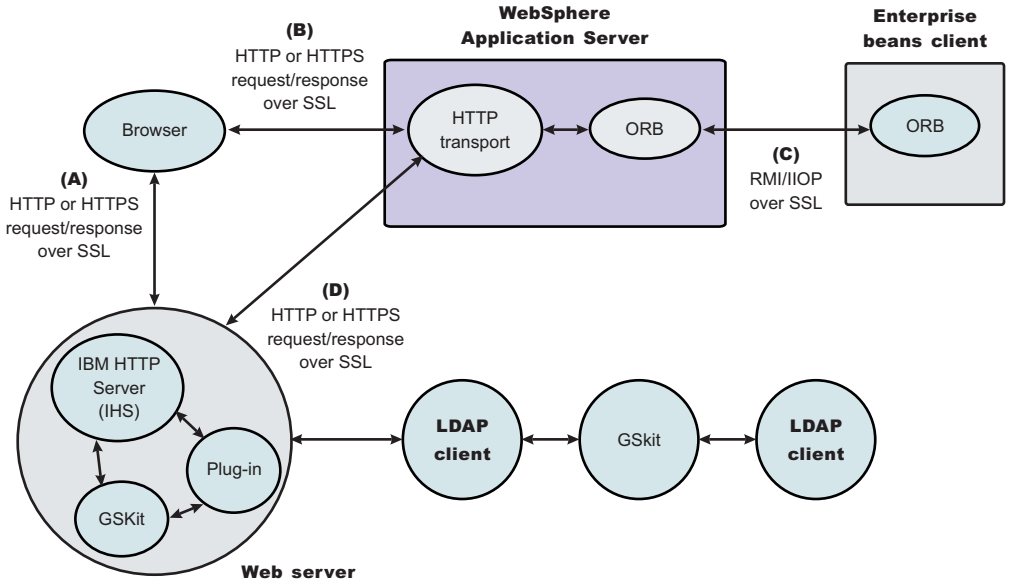


Figure 10. SSL and WebSphere Application Server

In this figure:
- The built-in HTTP transport in WebSphere Application Server accepts HTTP requests over SSL from a Web client like a browser.
- The Object Request Broker used in WebSphere Application Server can perform Internet Inter-ORB Protocol (IIOP) over SSL to secure the message.
- The secure LDAP client uses LDAP over SSL to securely connect to an LDAP user registry and is present only when LDAP is configured as the user registry.

WebSphere Application Server and the IBM Java Secure Socket Extension (IBMJSSE and IBMJSSE2) providers

Configuring the JSSE provider is very similar to configuring most other SSL implementations (for example, GSKit); however, a couple of differences are worth noting.
- The JSSE provider supports both signer and personal certificate storage in an SSL key file, but it also supports a separate file called a *trust file*. A trust file can contain only signer certificates. You can put all of your personal certificates in an SSL keyfile and your signer certificates in a trustfile. This support might be helpful, for example, if you have an inexpensive hardware cryptographic device with only enough memory to hold a personal certificate. In this case, the keyfile refers to the hardware device and the trustfile refers to a file on a disk that contains all of the signer certificates.
- The JSSE provider does not recognize the proprietary SSL keyfile format, which is used by the plug-in (*.kdb* files). Instead, the JSSE provider recognizes standard file formats such as Java Key Standard (JKS). SSL keyfiles might not be shared between the plug-in and application server. Furthermore, a different implementation of the key management utility must be used to manage application server key and trustfiles.

Certain limitations exist with the Java Secure Socket Extension (JSSE) provider:
- Customer code using JSSE and Java Cryptography Extension (JCE) APIs must reside within WebSphere Application Server environment. This restriction includes applications that are deployed in WebSphere Application Server and client applications in the J2EE application client environment.
- Hardware token support is limited to the Java Cryptography Extension V1.2.1, Hardware Cryptography IBMJCE4758
- The SSL protocol of Version 2.0 is not supported. In addition, the JSSE and JCE APIs are not supported with Java applet applications.

WebSphere Application Server and the Federal Information Processing Standards for Java Secure Socket Extension and Java Cryptography Extension providers

The Federal Information Processing Standards (FIPS)-approved Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE) providers are optional in WebSphere Application Server. By default, the FIPS-approved JSSE and JCE providers are disabled. When these providers are enabled, WebSphere Application Server uses FIPS-approved cryptographic algorithms in the IBMJCEFIPS provider package only.

Important: The IBMJCEFIPS module is a FIPS 140-2-approved cryptographic provider. For more information on the FIPS certification process, refer to Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

Authenticity:

Authenticity of client and server identities during a Secure Sockets Layer (SSL) connection is validated by both communicating parties using public key cryptography or asymmetric cryptography, to prove the claimed identity from each other.

Public key cryptography is a cryptographic method that uses public and private keys to encrypt and decrypt messages. The public key is distributed as a public key certificate while the private key is kept private. The public key is also a cryptographic inverse of the private key. Well known public key cryptographic algorithms such as the Rivest Shamir Adleman (RSA) algorithm and Diffie-Hellman (DH) algorithm are supported in WebSphere Application Server.

Public key certificates are either issued by a trusted organization like a certificate authority (CA) or extracted from a self-signed personal certificate by using the IBM Key Management Tool (iKeyman). A self-signed certificate is less secure and is not recommended for use in a production environment.

The public key certificate includes the following information:
- Issuer of the certificate
- Expiration date
- Subject that the certificate represents
- Public key belonging to the subject
- Signature by the issuer

You can link multiple key certificates into a certificate chain. In a certificate chain, the client is always first, while the certificate for a root CA is last. In between, each certificate belongs to the authority that issued the previous one.

During the Secure Sockets Layer (SSL) connection, a digital signature is also applied to avoid forged keys. The digital signature is an encrypted hash and cannot be reversed. It is very useful for validating the public keys.

SSL supports reciprocal authentication between the client and the server. This process is optional during the handshake. By default, a WebSphere Application Server client always authenticates its server during the SSL connection. For further protection, you can configure a WebSphere Application Server for client authentication.

Refer to the Transport Layer Security (TLS) specification at http://www.ietf.org/rfc/rfc2246.txt for further information.

Confidentiality:

Secure Sockets Layer (SSL) uses private or secret key cryptography or symmetric cryptography to support message confidentiality or privacy. After an initial handshake (a negotiation process by message

exchange), the client and server decide on a secret key and a cipher suite. Between the communicating parties, each message encryption and decryption using the secret key occurs based on the cipher suite.

Private key cryptography requires the two communicating parties to use the same key for encryption and decryption. Both parties must have the key and keep the key private. Well known secret key cryptographic algorithms include the Data Encryption Standard (DES), triple-strength DES (3DES), and Rivest Cipher 4 (RC4), which are all supported in WebSphere Application Server. These algorithms provide excellent security and quick encryption.

A cryptographic algorithm is a *cipher*, while a set of ciphers is a *cipher suite*. A cipher suite is a combination of cryptographic parameters that define the security algorithms and the key sizes used for authentication, key agreement, encryption strength, and integrity protection.

- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_AES_256_CBC_SHA
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_AES_256_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_AES_256_CBC_SHA
- SSL_DHE_DSS_WITH_RC4_128_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA

Important: Although anonymous cipher suites are enabled, the IBM version of the Java Secure Sockets Extension (JSSE) client trust manager does not support anonymous cipher suites. The default implementation can be overwritten by providing your own trust manager that does support anonymous cipher suites.

All of the previously mentioned cipher suites provide data integrity protection by using hash algorithms like `MD5` and `SHA-1`. The cipher suite names ending with `_SHA` indicate that the `SHA-1` algorithm is used. SHA-1 is considered a stronger hash, while MD5 provides better performance.

The `SSL_DH_anon_xxx` cipher suites (for example, those cipher suites that begin with `SSL_DH_anon_`, where, `anon` is *anonymous*) are not enabled on the product client side. Because the Java Secure Socket Extension (JSSE) client trust manager does not support anonymous connections, the JSSE client must always establish trust in the server. However, the `SSL_DH_anon_xxx` cipher suites are enabled on the server side to support another type of client connection. That client might not require trust in the server. These cipher suites are vulnerable to *man-in-the-middle* attacks and are strongly discouraged. In a *man-in-the-middle* attack, an attacker can intercept and potentially modify communications between two parties without either party being aware of the attack.

Where:

Name	Description
SSL	Secure Sockets Layer
RSA	• Public key algorithm developed by Rivest, Shamir and Adleman • Requires RSA or DSS key exchange
DH	• Diffie-Hellman public key algorithm • Server certificate contains the Diffie-Hellman parameters that are signed by the certificate authority (CA)
DHE	• Ephemeral Diffie-Hellman public key algorithm • Diffie-Hellman parameters are signed by a DSS or an RSA certificate, which is signed by the certificate authority (CA)
DSS	Digital Signature Standard, using the Digital Signature Algorithm for digital signatures
DES	• Data Encryption Standard, an symmetric encryption algorithm • Block cipher • Performance cost is high when using software without the support of a hardware cryptographic device
3DES	• Triple DES, increasing the security of DES by encrypting three times with different keys • Strongest of the ciphers • Performance cost is very high when using software without the support of a hardware cryptographic device support
RC4	• A stream cipher designed for RSA • Variable key-size stream cipher with key length from 40 bits to 128 bits
EDE	Encrypt-decrypt-encrypt for the triple DES algorithm
CBC	• Cipher block chaining • A mode in which every plain text block that is encrypted with the block cipher is first exclusive-ORed with the previous ciphertext block
128	128-bit key size
40	40-bit key size
EXPORT	Exportable
MD5	• Secure hashing function that converts an arbitrarily long data stream into a digest of fixed size • Produces 128-bit hash
SHA	• Secure Hash Algorithm, same as SHA-1 • Produces 160-bit hash
anon	For anonymous connections
NULL	No encryption
WITH	The cryptographic algorithm is defined after this key word

Refer to the Transport Layer Security (TLS) specification at http://www.ietf.org/rfc/rfc2246.txt for further information.

Integrity:

Secure Sockets Layer (SSL) uses a cryptographic hash function similar to checksum, to ensure data integrity in transit. Use the cryptographic hash function to detect accidental alterations in the data. This function does not require a cryptographic key. After a cryptographic hash is created, the hash is encrypted with a secret key. The private key belonging to the sender encrypts the hash for the digital signature of the message.

When secret key information is included with the cryptographic hash, the resulting hash is known as a *Key-Hashing Message Authentication Code* (HMAC) value. HMAC is a mechanism for message authentication that uses cryptographic hash functions. Use this mechanism with any iterative cryptographic hash function, in combination with a secret shared key.

In the product, both well known *one-way* hash algorithms, MD5 and SHA-1, are supported. One-way hash is an algorithm that converts processing data into a string of bits known as a *hash value* or a *message digest*. *One-way* means that it is extremely difficult to turn the fixed string back into the original data. The following explanation includes both the MD5 and SHA-1 *one-way* hash algorithms:

- MD5 is a hash algorithm designed for a 32-bit machine. It takes a message of arbitrary length as input and produces a 128-bit hash value as output. Although this process is less secure than SHA-1, MD5 provides better performance.
- SHA-1 is a secure hash algorithm specified in the Secure Hash Standard. It is designed to produce a 160-bit hash. Although it is slightly slower than MD5, the larger message digest makes it more secure against attacks like *brute-force collision*.

Refer to the Transport Layer Security (TLS) specification at http://www.ietf.org/rfc/rfc2246.txt for further information.

Configuring Secure Sockets Layer

Secure Sockets Layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy. The following is a listing of these components:

- Built-in HTTP Transport
- Object Request Broker (ORB) for client and Internet InterORB Protocol (IIOP)
- Secure Lightweight Directory Access Protocol (LDAP) client.

Configuring SSL is different between client and server with WebSphere Application Server and for Java Secure Socket Extension (JSSE) and System SSL.

1. Configure the client (JSSE). Use the `sas.client.props` file located, by default, in the *`install_root`*`/profiles/`*`profile_name`*`/properties` directory. The `sas.client.props` file is a configuration file that contains lists of property-value pairs, using the syntax *<property> = <value>*. The property names are case sensitive, but the values are not; the values are converted to lowercase when the file is read. Specify the following properties for an SSL connection:
   - com.ibm.ssl.protocol
   - com.ibm.ssl.keyStoreType
   - com.ibm.ssl.keyStore
   - com.ibm.ssl.keyStorePassword
   - com.ibm.ssl.trustStoreType
   - com.ibm.ssl.trustStore
   - com.ibm.ssl.trustStorePassword
   - com.ibm.ssl.enabledCipherSuites
   - com.ibm.ssl.contextProvider
   - com.ibm.ssl.keyStoreServerAlias
   - com.ibm.ssl.keyStoreClientAlias
2. Configure the client (System SSL).

   Configurations using System SSL are differentiated by z/OS Secure Authentication Services (z/SAS) and Common Secure Interoperability Version 2 (CSIv2) protocols. The z/SAS protocols use renamed

legacy environment variables that are provided by WebSphere Application Server for z/OS ,Version 6. CSIv2 uses a new properties file that is specified by a Java property and can be used by Java clients only.

- **z/SAS**:

  Note: z/SAS has been deprecated in WebSphere Application Server Version 6.

  a. Create an environment file for the client. Set the variables in the file as listed.
  b. Specify the SSL key ring through the `security_sslKeyring` variable to a key ring that is created for the client.
  c. Specify a user ID and password if using z/SAS basic authentication through the `client_protocol_user` and the `client_protocol_password` variables.
  d. Point to the environment file using the fully qualified path name through the environment variable `WAS_CONFIG_FILE`. For example, in the test shell script `test.sh`, export `WAS_CONFIG_FILE=/WebSphere/V5R0M0/AppServer/bin/current.env`.

- **CSIv2**: CSIv2 only supports Java clients and the Java com.ibm.CORBA.ConfigURL property must be specified to point to a properties file. You can specify individual properties on the Java invocation.

  a. Create or update the CSIv2 properties file with the properties.
  b. Specify the SSL key ring using `com.ibm.CSI.performSSL.Keyring`
  c. If using the Generic Security Service Username Password(GSSUP) authentication mechanism, specify the user ID and password using the `com.ibm.CSI.Rem.Userid` and `com.ibm.CSI.Rem.Password` property. Specify GSSUP using `com.ibm.COBRA.authenticationTarget=BasicAuth`, `com.ibm.CSI.performClientAuthenticationRequired`, `com.ibm.CSI.performClientAuthenticationRequired` and `com.ibm.CSI.performTransportAssocSSLTLSSupported`.
  d. If client certificate authentication is desired, specify: `com.ibm.CSI.performTLClientAuthenticationRequired` and `com.ibm.CSI.performTLClientAuthenticationSupported`.
  e. Specify the fully qualified path name of the properties file on the Java invocation. `-Dcom.ibm.CORBA.ConfigURL=file:/WebSphere/V5R0M0/AppServer/bin/CSI.properties`

3. Configure the server. Use the administrative console to configure an application server that makes SSL connections. To start the administrative console, specify the following Web address: http://*server_hostname*:9060/ibm/console.
4. Create a System SSL or JSSE repertoire. The type of repertoire depends on which function is configured. In general, you need to create both kinds of repertoires. System SSL repertoires are required to use SSL over HTTP and IIOP. A Java Secure Socket Extension (JSSE) repertoire is used to connect Simple Object Access Protocol (SOAP) connectors.

Configuring Secure Sockets Layer for Web client authentication:

To enable client-side certificate-based authentication, you must modify the authentication method that is defined on the Java 2 Platform, Enterprise Edition (J2EE) Web module that you want to manage. The Web module might already be configured to use the basic challenge authentication method. In this case, modify the challenge type to `client certificate`. This functionality is delivered to the WebSphere Application Server administrator in assembly tools. However, developers can use the Rational Web Developer environment to achieve the same result.

1. Launch the assembly tools. This step can be done either before an enterprise application archive `.ear` file is deployed into WebSphere Application Server or after deployment into the product. The latter option is discouraged in a production environment because it involves opening the expanded archive correlating to the enterprise application archive, found in the `installedApps` directory.
2. Locate and expand the Web module package under an application to enable the client-side certificate authentication method.

3. Select the appropriate Web application, and switch to the **Advanced** tab. Modify the authentication method to client certificate. The realm name is the scope of the login operation and is the same for all participating resources.
4. Click **OK**, and save the changes you made with the assembly tools.
5. Stop and restart the associated application server containing the resource, so that the security modification is included in the run time. Complete this action if the modification is made to a resource that already is deployed in WebSphere Application Server.

Now your enterprise application prompts the user for proof of identity with a certificate.

Note:

The Web server must also be configured to request a client certificate. If the Web server is external, refer to the appropriate configuration documentation. If the Web server is the Web container transport (for example, 9043) within WebSphere Application Server, verify that the **client authentication** flag is selected in the referenced SSL configuration.

Also, add the browser's signer certificate to the application server's keystore. For a self-signed personal certificate, the signer certificate is the is the public key of the personal certificate. For a certificate authority-signed server personal certificate, the signer certificate is the root certificate authority certificate of the certificate authority that signed the personal certificate.

Configuring Secure Sockets Layer for the Lightweight Directory Access Protocol client:

This topic describes how to establish a Secure Sockets Layer (SSL) connection between WebSphere Application Server and a Lightweight Directory Access Protocol (LDAP) server. This page provides an overview. Refer to the linked pages for more details. To understand SSL concepts, refer to "Secure Sockets Layer" on page 1179.

Setting up an SSL connection between WebSphere Application Server and an LDAP server requires the following steps:

1. Set up an LDAP server with users. The server configured in this example is IBM Directory Server. Other servers are configured differently. Refer to the documentation of the directory server you are using for details on SSL enablement. For a product-supported LDAP directory server, see the "Supported directory services" on page 971 article.
2. Configure certificates for the LDAP server using the key management utility (iKeyman) that is located in the *install_dir*/java/jre/bin directory.
3. Click **Key Database File > New**.
4. Type LDAPkey.kdb as the file name and a proper path and click **OK**.
5. Specify a password, confirm the password, and click **OK**.
6. Under Key database content, select **Personal Certificates**.
7. Click **New Self-signed**. The **Create New Self-Signed Certificate** panel is displayed. Type the following required information in the fields and click **OK**:
   Key Label
   LDAP_Cert
   Version
   Select the version of the X.509 certificate.
   Key size
   Select either a 512 or a 1024 bit size for your key.
   Common Name
   droplet.austin.ibm.com

   This common name is the host name where the WebSphere Application Server plug-in runs.

Organization
ibm
Country
US
Validity period
Specify the number days in which your certificate is valid.

8. Return to the Personal Certificates panel and click **Extract Certificate**.
9. Click the **Base64-encoded ASCII data** data type. Type LDAP_cert.arm as the file name and a proper path. Click **OK**.
10. Enable SSL on the LDAP server:
    a. Copy the LDAPkey.kdb, LDAPkey.sth, LDAPkey.rdb, and LDAPkey.crl files created previously to the LDAP server system, for example, the /Program Files/IBM/LDAP/ssl/ directory.
    b. Open the LDAP Web administrator from a browser (http://secnt3.austin.ibm.com/ldap, for example). IBM HTTP Server is running on secnt3.
    c. Click **SSL properties** to open the SSL Settings window.
    d. Click **SSL On** > **Server Authentication** and type an SSL port (636, for example) and a full path to the LDAPkey.kdb file.
    e. Click **Apply**, and restart the LDAP server.
11. Manage certificates for WebSphere Application Server using the default SSL key files.
    a. Open the *install_root*/profiles/default/etc/DummyServerTrustFile.jks file using the key management utility that shipped with WebSphere Application Server. The password is WebAS.

       Attention: It is recommended that you create your own trustfile.jks file rather than modifying the DummyServerTrustFile.jks file. If you use the DummyServerTrustFile.jks file, there is a risk that your changed settings might be overwritten if you update the product with iFixes.
    b. Click **Signer certificate > Add**. The Add CA's Certificate from a File window is displayed. Specify LDAP_cert.arm for the file name. Complete this step for all the servers and the deployment manager.
12. Establish a connection between the WebSphere Application Server and the LDAP server using the WebSphere Application Server administrative console.
    a. Click **Security > Global security**.
    b. Under User registries, click **LDAP**.
    c. Enter the **Server ID**, **Server Password**, **Type**, **Host**, **Port**, and **Base Distinguished Name** fields.
    d. Select the **SSL Enabled** option. The port is the same port number that the LDAP server is using for SSL (636, for example).
    e. Click **Apply**.
    f. Return to the Global security panel and click **Authentication Mechanisms > LTPA > Single SignOn (SSO)**.
    g. Under Additional properties, click **Single signon (SSO)**.
    h. Type in a domain name (austin.ibm.com, for example).
    i. Click **Apply**.
13. Enable global security.
    a. Click **Security > Global Security**.
    b. Select the **Enable global security** option.
    c. Select the **Lightweight Third Party Authentication (LTPA)** option as the active authentication mechanism and the **Lightweight Directory Access Protocol (LDAP) user registry** option as the active user registry.

       Note: Verify that the security level for the LDAP server is set to HIGH. The default security level is HIGH (128-bit).

d. Click **Apply** and **Save**.
e. Verify that the ibm-slapdSSLCipherSpecs parameter in the *LDAP_install_root*/etc/slapd32.conf file has the value, 15360, instead of 12288.
f. Restart the servers.

Restarting the servers ensures that the security settings are synchronized between the deployment manager and the application servers.

You can test the configuration by accessing https://*fully_qualified_host_name*:9443/snoop. You are presented with a login challenge. This test can be beneficial when using LDAP as your user registry. Sensitive information can flow between the WebSphere Application Server and the LDAP server, including passwords. Using SSL to encrypt the data protects this sensitive information.

1. If you are enabling security, make sure that you complete the remaining steps. As the final step, validate this configuration by clicking **OK** or **Apply** in the Global Security panel. Refer to the "Configuring global security" on page 881 article for detailed steps on enabling global security.
2. For changes in this panel to become effective, save, stop, and start all WebSphere Application Servers (cells, nodes and all the application servers).
3. After the server starts up, go through all the security-related tasks (getting users, getting groups, and so on) to make sure that the changes to the filters are functioning.

Changing the default Secure Sockets Layer repertoire key files:

If you modify the default digital certificates in the key rings belonging to the node agent and the deployment managers or application servers, you must verify that the public certificate of the new certificate authority is added as a trust certificate in the key rings of all servers to which it needs to communicate. This action includes modifying the certificates so they are issued from a different certificate authority (for example, if you use a commercial certificate authority).

Within a given cell, the:

- Deployment manager and node agents must be able communicate
- Node agents must be able to communicate to all servers within the node

If you modify the repertoire definitions, you must update the:

- System SSL repertoire used by HTTP
- System SSL repertoire used for Internet InterORB Protocol (IIOP) communications
- Java Secure Socket Extension (JSSE) repertoire that is used for the Simple Object Access Protocol (SOAP) and Java Management Extensions (JMX) connector, if applicable

Configuring Secure Sockets Layer for Java client authentication:

WebSphere Application Server supports Java client authentication using a digital certificate when the client attempts to make a Secure Sockets Layer (SSL) connection. The authentication occurs during an SSL handshake. The SSL handshake is a series of messages exchanged over the SSL protocol to negotiate for connection-specific protection. During the handshake, the secure server requests that the client to send back a certificate or certificate chain for the authentication.

To configure SSL for Java client authentication, consider the following questions:

- Have you enabled security with your WebSphere Application Server? For more information, see "Configuring global security" on page 898.
- Have you configured your server to support secure transport for the inbound z/SAS (on the z/OS platform) or CSI authentication protocol?
- Have you configured your server to support client authentication at the transport layer for the inbound zSAS (on the z/OS platform) or CSI authentication protocol?
- If you are using a self-signed personal certificate, have you exported the public certificate from the Service Access Facility (SAF)?

- If you are using a certificate authority (CA)-signed personal certificate, have you received the root certificate of the CA?
- If you are using a self-signed personal certificate, have you imported the public certificate into SAF as a signer certificate?
- If you are using a CA-signed (certificate authority) personal certificate, have you imported the CA root certificate into your target Java truststore file as a signer certificate?
- Does the common name (CN) specified in your personal certificate name exist in your configured user registry or is there a SAF mapping for the certificate?

If you answer yes to all of these questions that are appropriate to your product and platform, you can configure SSL for Java client authentication.

1. "Configuring Common Secure Interoperability Version 2 for Secure Sockets Layer client authentication."
2. "Adding keystore files."
3. "Adding truststore files" on page 1190.
4. Save changes.
5. Restart the server if you configured the server.

A secure client connects to a secure Internet InterORB Protocol (IIOP) server that requires client authentication at the transport layer. If a connection problem occurs, you can set a Java property, `javax.net.debug=true`, before you run your client or your server to generate debugging information. See Troubleshooting security configurations for further information about how to debug an IBMJSSE problem.

Configuring Common Secure Interoperability Version 2 for Secure Sockets Layer client authentication:

Configure the Secure Sockets Layer (SSL) client authentication using the `sas.client.props` configuration file or the administrative console. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` file is located in the *install_root*`/profiles/`*profile_name*`/properties` directory of your WebSphere Application Server installation.

To configure a WebSphere Application Server, use the administrative console. To start the administrative console, specify URL: `http://`*server host_name*`:9060/ibm/console`.

To configure a Java client application, complete the following steps, which explain how to edit the `sas.client.props` file directly:

1. To require SSL client authentication, set property `com.ibm.CSI.performTLClientAuthenticationRequired=true`. Do not set this property unless you know your target server also supports SSL client authentication for the inbound CSI authentication protocol.
2. To support SSL client authentication, set the property `com.ibm.CSI.performTLClientAuthenticationSupported=true`.
3. Specify the com.ibm.CORBA.ConfigURL property with the fully qualified path of your Java property file when you run your application. For example, `-Dcom.ibm.CORBA.ConfigURL=file:/WebSphere/AppServer/profiles/`*profile_name*`/properties/sas.client.props`

Adding keystore files:

A keystore contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. In WebSphere Application Server, adding keystore files to the configuration is different between client and server. For the client, a keystore file is added to a property file like `sas.client.props`. For the server, a keystore file is added through the WebSphere Application Server administrative console.

Before you add the keystore file to your configuration, consider the following questions:

- Is a self-signed or a certificate authority (CA)-signed personal certificate created in the keystore file?
- If you configure client authentication using digital certificates, is the public key of the signed personal certificate imported as a signer certificate into the server truststore file?

1. Add a keystore file into a client configuration by editing the `sas.client.props` file and setting the following properties:
   - **com.ibm.ssl.keyStoreType** for the keystore format. Range: JKS (default), PKCS12, JCEK, JCERACFKS.
   - **com.ibm.ssl.keyStore** for a fully qualified path to the keystore file. The keystore file contains private keys and sometimes public keys.

     For RACF key rings, set com.ibm.ssl.keyStore to `safkeyring:///`.
   - **com.ibm.ssl.keyStorePassword** for the password to access the keystore file.

     For RACF key rings, set com.ibm.ssl.keyStorePassword to `password`, and set com.ibm.ssl.keyStoreType to JCERACFKS, if using a RACF key ring.
2. Add a keystore file into a server configuration:
   a. Start the administrative console by specifying: http://*server_hostname*:9060/ibm/console.
   b. Click **Security > SSL**.
   c. Optional: Click **New SSSL repertoire** to create a new Secure Sockets Layer (SSL) setting alias if one does not exist or click **New JSSE repertoire** to create a new Java Secure Sockets Extension (JSSE) repertoire.
   d. Select the alias that you want to add into the keystore file.
   e. Type in the key file name for the path of the keystore file.

      Type `safkeyring:///`*`your_keyring_name`* if you want to use certificates and keys that are contained in a RACF key ring.
   f. Type in the key file password for the password to access the keystore file.

      Type password if you are using a RACF key ring.
   g. Select the key file format for the keystore type. Range: JKS (default), PKCS12, or JCEK.
   h. Click **OK** and **Save** to save the configuration.

The SSL configuration alias now has a valid keystore file for an SSL connection.
- SSL connection for Internet InterORB Protocol (IIOP)
- SSL connection for Lightweight Directory Access Protocol (LDAP)
- SSL connection for Hypertext Transfer Protocol (HTTP)

Adding truststore files:

A *truststore file* is a key database file that contains public keys. The public key is stored as a signer certificate. The keys are used for a variety of purposes, including authentication and data integrity. In WebSphere Application Server, adding truststore files to the configuration is different between client and server. For the client, a truststore file is added to a property file, like `sas.client.props`. For the server, a truststore file is added through the WebSphere Application Server administrative console.

Before you add the truststore file to your configuration, ask the following questions:
- If you configure for client authentication using digital certificate, has the public key of the client personal certificate been imported as a signer certificate into the server truststore file?
- Does the truststore file contain all the required signer certificates with respect to the keystore files of the target servers?

1. Add a truststore file into a client configuration, by editing the `sas.client.props` file and setting the following properties:
   - **com.ibm.ssl.trustStoreType** for the truststore format. Range: JKS (default), PKCS12, JCEK, JCERACFKS.

     Use JCERACFKS if you are using a RACF key ring as the trust store.

- **com.ibm.ssl.trustStore** for the name of the RACF key ring that you want Java Secure Socket Extension (JSSE) to use. Specify `safkeyring:///`.
- **com.ibm.ssl.trustStorePassword** for the password to access the truststore file.

  Set the com.ibm.ssl.trustStorePassword property to `password` if you are using a RACF key ring as a trust store.

2. Add a truststore file into a server configuration:
   a. Start the administrative console by specifying : http://*server_host_name*:9060/ibm/console
   b. Click **Security > SSL**.
   c. Create a new Secure Sockets Layer (SSL) setting alias if one does not exist.
   d. Select the alias that you want to add into the truststore file.
   e. Type the trust file name for the path of the truststore file. Type `safkeyring:///` if you are using a RACF key ring for the trust store.
   f. Type the trust file password for the password to access the truststore file. Type `password` if you are using a RACF key ring for the trust store.
   g. Select the trust file format for the truststore type. JKS (Default), PKCS12, JCEK.
   h. Click **OK** and **Save** to save the configuration.

The SSL configuration alias now contains a valid truststore file for an SSL connection.
- SSL connection for Internet InterORB Protocol (IIOP)
- SSL connection for Lightweight Directory Access Protocol (LDAP)
- SSL connection for Hypertext Transfer Protocol (HTTP)

Editing the sas.client.props file using the administrative console:

To edit the `sas.client.props` file using the administrative console, complete the following steps:

1. Start the administrative console.
2. Expand **Security > Global security**.
3. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**.
4. Select **Supported** or **Required** for Client certificate authentication.
5. Click **OK**.
6. If you selected **Required** in step 4, configure the CSIv2 outbound authentication as well to support the client certificate authentication. Otherwise, you can skip this step. Return to the Global security panel and under Authentication, click **CSIv2 Outbound Authentication**. Select either **Supported** or **Required** for Client certificate authentication.
7. Click **CSIv2 Outbound Transport**.
8. Select an SSL setting from the SSLSettings list for keystore, truststore, cryptographic token, SSL protocol, and ciphers use.
9. Create an alias from the SSL Configuration Repertoires panel for an SSL setting.
10. Update the SSL setting selected in CSIv2 Inbound Transport accordingly.
11. Save your configuration.
12. Restart the server for the changes to become effective.

Client authentication using digital certificates is performed during SSL connection. A secure client connects using SSL to a secure Internet InterORB Protocol (IIOP) server with client authentication at the transport layer.

Secure Sockets Layer configuration repertoire settings:

Use this page to define a new Secure Sockets Layer (SSL) alias. Using the SSL configuration repertoire, administrators can define any number of SSL settings to use in configuring the Hypertext Transfer Protocol with SSL (HTTPS), Internet InterORB Protocol with SSL (IIOPS) or Lightweight Directory Access Protocol

with SSL (LDAPS) connections. You can pick one of the SSL settings defined here from any location within the administrative console that supports SSL connections. This flexibility simplifies the SSL configuration process because you can reuse many of these SSL configurations by specifying the alias in multiple places.

To view this administrative console page, click **Security** > **SSL**.

Click **New** to create a new SSL Configuration Repertoire alias.

Click **Delete** to remove an SSL Configuration Repertoire alias. If an SSL configuration alias is referenced in the configuration and is deleted here, then an SSL connection fails when the deleted alias is accessed.

Note: If you are choosing to create a new SSL repertoire the type can be either of the following:

- Java Secure Socket Extension (JSSE) for JMX SOAP Connector
- System SSL (SSSL)for Web container and ORB transport

Alias:

Specifies the name of the specific SSL setting.

On the WebSphere Application Server Network Deployment product installation, the default cell SSL alias is used for the HTTPS transport when creating a new server.

Type:

Specifies the type of repertoire configured for the alias listed.

The value is either SSSL for System Secure Sockets Layer repertoire or JSSE for Java Secure Sockets Extension repertoire.

6.1+

JSSE Defines an SSL configuration that Java Secure Socket Extensions (JSSE) can use

SSSL Defines an SSL configuration that z/OS System SSL can use. Use this configuration to process SSL sessions using the following methods:

- Common Secure Interoperability version 2 (CSIv2)
- z/OS Secure Authentication Services (zSAS) protocols
- Web container inbound requests

Repertoire settings:

Use this page to configure Secure Sockets Layer (SSL) or Java Secure Sockets Extension (JSSE) settings for the server. To configure Secure Sockets Layer (SSL), you need to define an SSL configuration repertoire. A repertoire contains the details necessary for building an SSL connection, such as the location of the key files, their type and the available ciphers. WebSphere Application Server provides a default repertoire called DefaultSSLSettings.

To view this administrative console page, click **Security > SSL >***alias_name*.

Alias:

Specifies the name of the specific SSL setting

Data type: String

This field is used on the System SSL Repertoire and Java Secure Sockets Extension (JSSE) Repertoire panels.

Key file name:

Specifies the fully qualified path to the SSL key file that contains public keys and might contain private keys.

On z/OS, there are two types of Secure Sockets Layer (SSL): Java Secure Socket Extension (JSSE) SSL and System SSL. For Java Secure Socket Extension (JSSE) SSL, the key file name specifies the fully qualified path to the SSL key file that contains public keys and private keys. For System SSL, the key file name specifies the name of the System Authorization Facility (SAF) key ring. The key file name might also be the name of the SAF key ring that contains public and private keys.

For JSSE SSL, the key file specifies the `keystore` file. The key file might also specify the System Authorization Facility (SAF) Key ring that contains certificates and keys. You can create a JSSE SSL keystore file by using the keytool utility found in the WebSphere `bin` directory. The key file contains certificates and keys.

For System SSL or JSSE, you can create an SSL key ring by using the Resource Access Control Facility (RACF) command, RACDCERT. Issue this command in your MVS environment, such as TSO READY or ISPF option 6. The key ring contains the private certificate of this server and certificates of trusted certificate authorities. The certificates for the trusted certificate authorities validate the client certificates and other server certificates that are exchanged with this server during the SSL handshake. The repertoires that you define for a server require identical key file names.

Data type: String

This field is used on the System SSL Repertoire and JSSE Repertoire panels.

Client authentication:

Specifies whether to request a certificate from the client for authentication purposes when making a connection.

When performing client authentication with the Internet InterORB Protocol (IIOP) for EJB requests, click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication** or **Authentication protocol > CSIv2 outbound authentication**. Select the appropriate option under Client certificate authentication.

Default: Disabled
Range: Enabled or Disabled

This field is used on the System SSL Repertoire and JSSE Repertoire panels.

Security level:

Specifies whether the server selects from a preconfigured set of security levels.

Data type:	Valid values include Low, Medium or High. • Low specifies digital signing ciphers only without encryption. • Medium specifies 40-bit ciphers only including digital signing. • High specifies 128-bit ciphers only including digital signing. To specify all ciphers or any particular range, you can set the com.ibm.ssl.enabledCipherSuites property. See the SSL documentation for more information.
Default:	High
Range:	Low, Medium, or High

Note: The Simple Object Access Protocol (SOAP) connector does not use security level.

This field is used on the System SSL Repertoire and JSSE Repertoire panels.

V3 timeout:

Specifies the length of time that a browser can reuse a System SSL Version 3 session ID without renegotiating encryption keys with the server.

The repertoires that you define for a server require the same V3 timeout value.

Data type	integer
Default	100
Range	1 to 86400

This field is used on the System SSL Repertoire panel.

Cipher suites:

Specifies a list of supported cipher suites that can be selected during the SSL handshake. If you select cipher suites individually here, you override the cipher suites set in the Security Level field.

Data type:	String
Default:	None

Note: The SOAP connector does not use cipher suites.

This field is used on the System SSL Repertoire and JSSE Repertoire panels.

Provider:

Refers to a package that implements a subset of the Java security application programming interface (API) cryptography aspects.

If you select **Predefined JSSE provider**, select a provider from the menu.

WebSphere Application Server has the IBMJSSE predefined provider.

The name for the Cipher suite property is `com.ibm.ssl.enabledCiphersuites`. The name for the protocol property is `com.ibm.ssl.protocol`.

This field is used on the JSSE Repertoire panel.

Protocol:

Specifies which SSL protocol to use.

Default SSLv3
Range SSL, SSLv2, SSLv3, TLS, TLSv1

This field is used on the JSSE Repertoire panel.

Key file password:

Specifies the password for accessing the SSL key file.

Data type: String

This field is used on the JSSE Repertoire panel.

Key file format:

Specifies the format of the SSL key file.

You can choose from the following key file formats: JKS, JCEK, PKCS12. The JKS format does not store a shared key. For more secure key files, use the JCEK format. PKCS12 is the standard file format.

Data type: String
Default: JKS
Range: JKS, PKCS12, JCEK

This field is used on the JSSE Repertoire panel.

Trust file name:

Specifies the fully qualified path to a trust file containing the public keys.

You can create a trust file by using the keytool utility located in the WebSphere *bin* directory.

Unlike the SSL key file, no personal certificates are referenced; only signer certificates are retrieved. The default SSL trust files, `DummyClientTrustFile.jks` and `DummyServerTrustFile.jks`, contain multiple test public keys as signer certificates that can expire. The following public keys expire on October 13, 2021:

- WebSphere Application Server Version 4.x test certificates
- WebSphere Application Server Version 5.x test certificates
- WebSphere Application Server CORBA C++ client
- WebSphere Application Server Version 6.0.x test certificates

The test certificates are only intended for use in a test environment.

If a trust file is not specified but the SSL key file is specified, then the SSL key file is used for retrieval of signer certificates as well as personal certificates.

Data type: String

This field is used on the JSSE Repertoire panel.

Trust file password:

Specifies the password for accessing the SSL trust file.

Data type: String

This field is used on the JSSE Repertoire panel.

Trust file format:

Specifies the format of the SSL trust file.

You can choose from the following trust file formats: JKS, JCEK, PKCS12. The JKS format does not store a shared key. For more secure key files, use the JCEK format. PKCS12 is the standard file format.

Data type: String
Default: JKS
Range: JKS, JCEK, PKCS12

This field is used on the JSSE Repertoire panel.

Secure Sockets Layer settings for custom properties:

Use this page to configure additional Secure Sockets Layer (SSL) settings for a defined alias.

To view this administrative console page, click **Security** > **SSL** > *alias_name* > **Custom properties**.

Custom Properties:

Specifies the name-value pairs that you can use to configure additional SSL settings beyond those available in the `com.ibm.ssl.protocol` administrative interface.

This value is the SSL protocol used (including its version). The possible values are SSL, SSLv2, SSLv3, TLS, or TLSv1. The default value, SSL, is backward-compatible with the other SSL protocols.

com.ibm.ssl.keyStoreProvider
The name of the key store provider to use. Specify one of the security providers listed in your `java.security` file, which has a keystore implementation. The default value is IBMJCE.

com.ibm.ssl.keyManager
The name of the key management algorithm to use. Specify any key management algorithm that is implemented by one of the security providers listed in your `java.security` file. The default value is IbmX509.

com.ibm.ssl.trustStoreProvider
The name of the trust store provider to use. Specify one of the security providers listed in your `java.security` file, which has a truststore implementation. The default value is IBMJCE.

com.ibm.ssl.trustManager
The name of the trust management algorithm to use. Specify any trust management algorithm that is implemented by one of the security providers listed in your `java.security` file. The default value is IbmX509.

com.ibm.ssl.trustStoreType
The type or format of the truststore file. The possible values are JKS, PKCS12, JCEK. The default value is JKS.

com.ibm.ssl.enabledCipherSuites
The list of cipher suites to enable. By default, this is not set and the set of cipher suites used is

determined by the value of the security level (high, medium, or low). A cipher suite is a combination of cryptographic algorithms used for an SSL connection. Enter a space-separated list of any of the following cipher suites:

- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

Data type: String

Cryptographic token:

Specifies information about the cryptographic tokens related to SSL support.

A cryptographic token is a hardware or software device that has a built-in keystore implementation. Document the exact values for the following fields found in the literature of your supported cryptographic device.

Configuring Federal Information Processing Standard Java Secure Socket Extension files:

In WebSphere Application Server Version 6, the Java Secure Socket Extension (JSSE) provider used is the IBMJSSE2 provider. This provider delegates encryption and signature functions to the Java Cryptography Extension (JCE) provider. Consequently, IBMJSSE2 does not need to be Federal Information Processing Standard (FIPS)-approved because it does not perform cryptography. However, the JCE provider requires FIPS-approval.

WebSphere Application Server provides a FIPS-approved IBMJCEFIPS provider that IBMJSSE2 can utilize. The IBMJCEFIPS provider shipped in WebSphere Application Server Version 6 supports the following SSL ciphers:

- SSL_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_AES_128_CBC_SHA

- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

Even though the IBMJSSEFIPS provider is still present, the runtime does not use this provider. If IBMJSSEFIPS is specified as a contextProvider, WebSphere Application Server automatically defaults to the IBMJSSE2 provider (with the IBMJCEFIPS provider) for supporting FIPS in Version 6. When enabling FIPS in the server Global Security Panel, the runtime always uses IBMJSSE2. despite whatever contextProvider you specify for SSL (IBMJSSE, IBMJSSE2 or IBMJSSEFIPS). Also, because FIPS requires the SSL protocol be TLS, the runtime always uses TLS when FIPS is enabled regardless of the SSL protocol setting in the SSL repertoire. This simplifies the FIPS configuration in Version 6 because an administrator only needs to enable the FIPS flag in the Global Security Panel to enable all transports using SSL.

1. Click **Security > Global Security**. Select the **Use the Federal Information Processing Standard (FIPS)** option and click **OK**. IBMJSSE2 and IBMJCEFIPS is enabled.
2. If you have a Java client that must access enterprise beans, change the `com.ibm.security.useFIPS` property value from `false` to `true` in the *install_dir*`/profiles/`*profile_name*`/properties/sas.client.props` file.
3. If you have an administrative client using the Simple Object Access Protocol (SOAP) connector, modify the `install_dir/profiles/profile_name/properties/soap.client.props` file on the administrative client and set the following property:

   ```
   #com.ibm.ssl.contextProvider=IBMJSSE2
   com.ibm.ssl.contextProvider=IBMJSSEFIPS
   ```

   Note: Note: Specifying IBMJSSEFIPS indicates that the client wants to be in FIPS mode, and the runtime uses the IBMJSSE2 provider in combination with the IBMJCEFIPS provider.

After completing these steps, a FIPS-approved JSSE or JCE provider offers increased encryption capabilities. However, when you use FIPS-approved providers:

- By default, Microsoft Internet Explorer Version 5.5 might not have Transport Layer Security (TLS) enabled. To enable TLS, open the Internet Explorer browser and click **Tools > Internet Options**. On the Advanced tab, select the Use TLS 1.0 option.

  Note: Netscape Version 4.7.x and earlier versions might not support TLS.
- IBM Directory Server Version 5.1 (and earlier versions) do not support TLS.
- If you have an administrative client that uses a SOAP connector and you enable FIPS, add the following lines to the *install_dir*`/profiles/`*profile_name*`/properties/soap.client.props` file:

  ```
  com.ibm.ssl.contextProvider=IBMJSSEFIPS
  ```
- When you select the **Use the Federal Information Processing Standard (FIPS)** option on the Global Security panel, the Lightweight Third-Party Authentication (LTPA) token format is not backwards-compatible with previous releases of WebSphere Application Server. However, you can continue to use the LTPA keys configured using a previous version of WebSphere Application Server.

Note: When enabling FIPS, you cannot configure cryptographic token devices in the SSL repertoires. IBMJSSE2 must use IBMJCEFIPS when utilizing cryptographic services for FIPS.

The following are the FIPS 140-2 approved cryptographic providers:

- IBMJCEFIPS (certificate 376)
- IBMJSSEFIPS (certificate 409)
- IBM Cryptography for C (ICC) (certificate 384)

The relevant certificates are listed on the NIST Web site: Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List

Digital certificates:

Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

A digital certificate is equivalent to an electronic ID card. It serves two purposes:
- Establishes the identity of the owner of the certificate
- Distributes the owner's public key

Certificates are issued by trusted parties, called *certificate authorities* (CAs). These authorities can be commercial ventures or they can be local entities, depending on the requirements of your application. Regardless, the CA is trusted to adequately authenticate users before issuing certificates. A CA issues certificates with digital signatures. When a user presents a certificate, the recipient of the certificate validates it by using the digital signature. If the digital signature validates the certificate, the certificate is recognized as intact and authentic. Participants in an application only need to validate certificates; they do not need to authenticate users. The fact that a user can present a valid certificate proves that the CA has authenticated the user. The descriptor, *trusted third-party*, indicates that the system relies on the trustworthiness of the CAs.

Contents of a digital certificate

A certificate contains several pieces of information, including information about the owner of the certificate and the issuing CA. Specifically, a certificate includes:
- The distinguished name (DN) of the owner. A DN is a unique identifier, a fully qualified name including not only the common name (CN) of the owner but the owner's organization and other distinguishing information.
- The public key of the owner.
- The date on which the certificate was issued.
- The date on which the certificate expires.
- The distinguished name of the issuing CA.
- The digital signature of the issuing CA. (The message-digest function is run over all the preceding fields.)

The core idea of a certificate is that a CA takes the owner's public key, signs the public key with its own private key, and returns the information to the owner as a certificate. When the owner distributes the certificate to another party, it signs the certificate with its private key. The receiver can extract the certificate (containing the CA signature) with the owner's public key. By using the CA public key and the CA signature on the extracted certificate, the receiver can validate the CA signature. If it is valid, the public key used to extract the certificate is recognized as good. The owner signature is then validated, and if the validation succeeds, the owner is successfully authenticated to the receiver.

The additional information in a certificate helps an application decide whether to honor the certificate. With the expiration date, the application can determine if the certificate is still valid. With the name of the issuing CA, the application can check that the CA is considered trustworthy by the site.

A process that uses certificates must provide its personal certificate, the one containing its public key, and the certificate of the CA that signed its certificate, called a *signer certificate*. In cases where chains of trust are established, several signer certificates can be involved.

Requesting certificates

To get a certificate, send a certificate request to the CA. The certificate request includes:
- The distinguished name of the owner (the user for whom the certificate is requested).
- The public key of the owner.
- The digital signature of the owner.

The message-digest function is run over all these fields.

The CA verifies the signature with the public key in the request to ensure that the request is intact and authentic. The CA then authenticates the owner. Exactly what the authentication consists of depends on a prior agreement between the CA and the requesting organization. If the owner in the request is successfully authenticated, the CA issues a certificate for that owner.

Using certificates: Chain of trust and self-signed certificate

To verify the digital signature on a certificate, you must have the public key of the issuing CA. Because public keys are distributed in certificates, you must have a certificate for the issuing CA that is signed by the issuer. One CA can certify other CAs, so a chain of CAs can issue certificates for other CAs, all of whose public keys you need. Eventually, you reach a root CA that issues itself a self-signed certificate. To validate a user's certificate, you need certificates for all intervening participants, back to the root CA. Then you have the public keys you need to validate each certificate, including the user's.

A self-signed certificate contains the public key of the issuer and is signed with the private key. The digital signature is validated like any other, and if the certificate is valid, the public key it contains is used to check the validity of other certificates issued by the CA. However, anyone can generate a self-signed certificate. In fact, you can probably generate self-signed certificates for testing purposes before installing production certificates. The fact that a self-signed certificate contains a valid public key does not mean that the issuer is really a trusted certificate authority. To ensure that self-signed certificates are generated by trusted CAs, such certificates must be distributed by secure means (hand-delivered on floppy disks, downloaded from secure sites, and so on).

Applications that use certificates store these certificates in a *keystore* file. This file typically contains the necessary personal certificates, its signing certificates, and its private key. The private key is used by the application to create digital signatures. Servers always have personal certificates in their keystore files. A client requires a personal certificate only if the client must authenticate to the server when mutual authentication is enabled.

To allow a client to authenticate to a server, a server keystore file contains the private key and the certificate of the server and the certificates of its CA. A client truststore file must contain the signer certificates of the CAs of each server to which the client must authenticate.

If mutual authentication is needed, the client keystore file must contain the client private key and certificate. The server truststore file requires a copy of the certificate of the client CA.

Digital signatures:

A *digital signature* is a number attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

This signature establishes the following information:
- The integrity of the message: Is the message intact? That is, has the message been modified between the time it was digitally signed and now?
- The identity of the signer of the message: Is the message authentic? That is, was the message actually signed by the user who claims to have signed it?

A digital signature is created in two steps. The first step distills the document into a large number. This number is the *digest code* or *fingerprint*. The digest code is then encrypted, resulting in the digital signature. The digital signature is appended to the document from which the digest code was generated.

Several options are available for generating the digest code. WebSphere Application Server supports the MD5 message digest function and the SHA1 secure hash algorithm, but these procedures reduce a message to a number. This process is not encryption, but a sophisticated checksum. The message cannot regenerate from the resulting digest code. The crucial aspect of distilling the document to a number is that if the message changes, even in a trivial way, a different digest code results. When the recipient gets a

message and verifies the digest code by recomputing it, any changes in the document result in a mismatch between the stated and the computed digest codes.

To stop someone from intercepting a message, changing it, recomputing the digest code, and retransmitting the modified message and code, you need a way to verify the digest code as well. To verify the digest code, reverse the use of the public and private keys. For private communication, it makes no sense to encrypt messages with your private key; these keys can be decrypted by anyone with your public key. This technique can be useful for proving that a message came from you. No one can create it because no one else has your private key. If some meaningful message results from decrypting a document by using someone's public key, the decryption process verifies that the holder of the corresponding private key did encrypt the message.

The second step in creating a digital signature takes advantage of this reverse application of public and private keys. After a digest code is computed for a document, the digest code is encrypted with the sender's private key. The result is the digital signature, which is attached to the end of the message.

When the message is received, the recipient follows these steps to verify the signature:
1. Recomputes the digest code for the message.
2. Decrypts the signature by using the sender's public key. This decryption yields the original digest code for the message.
3. Compares the original and recomputed digest codes. If these codes match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

Public key cryptography:

All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on what is called private key encryption. Only within the last 30 years has a challenge to private key encryption appeared - public key encryption.

Private key encryption

Private-key encryption systems use a single key that is shared between the sender and the receiver. Both must have the key; the sender encrypts the message by using the key, and the receiver decrypts the message with the same key. Both must keep the key private to keep their communication private. This kind of encryption has characteristics that make it unsuitable for widespread, general use:
- Private key encryption requires a key for every pair of individuals who need to communicate privately. The necessary number of keys rises dramatically as the number of participants increases.
- The fact that keys must be shared between pairs of communicators means the keys must somehow be distributed to the participants. The need to transmit secret keys makes them vulnerable to theft.
- Participants can communicate only by prior arrangement. There is no way to send a usable encrypted message to someone spontaneously. You and the other participant must make arrangements to communicate by sharing keys.

Private-key encryption is also called *symmetric encryption*, because the same key is used to encrypt and decrypt the message.

Public key encryption

Public key encryption uses a pair of mathematically related keys. A message encrypted with the first key must be decrypted with the second key, and a message encrypted with the second key must be decrypted with the first key.

Each participant in a public-key system has a pair of keys. The symmetric (private) key is kept secret. The other key is distributed to anyone who wants it; this key is the public key.

To send an encrypted message to you, the sender encrypts the message by using your public key. When you receive the message, you decrypt it by using your symmetric key. To send a message to someone, you encrypt the message by using the recipient's public key. The message can be decrypted with the recipient's symmetric key only. This kind of encryption has characteristics that make it very suitable for general use:

- Public-key encryption requires only two keys per participant. The increase in the total number of keys is less dramatic as the number of participants increases, compared to symmetric key encryption.
- The need for secrecy is more easily met. Only the symmetric key needs to be kept symmetric and because it does not need to be shared, the symmetric key is less vulnerable to theft in transmission than the shared key in a symmetric key system.
- Public keys can be published, which eliminates the need for prior sharing of a secret key before communication. Anyone who knows your public key can use it to send you a message that only you can read.

Public-key encryption is also called *asymmetric encryption*, because the same key cannot be used to encrypt and decrypt the message. Instead, one key of a pair is used to undo the work of the other. WebSphere Application Server uses the Rivest Shamir Adleman (RSA) public and symmetric key encryption algorithm.

With symmetric key encryption, you have to be careful of stolen or intercepted keys. In public-key encryption, where anyone can create a key pair and publish the public key, the challenge is in verifying that the owner of the public key is really the person you think it is. Nothing prevents a user from creating a key pair and publishing the public key under a false name. The listed owner of the public key cannot read messages encrypted with that key because the owner does not have the symmetric key. If the creator of the false public key can intercept these messages, that person can decrypt and read messages intended for someone else. To counteract the potential for forged keys, public-key systems provide mechanisms for validating public keys and other information with digital signatures and digital certificates.

Configuring to use cryptographic tokens

You can configure cryptographic token support in both client and server configurations. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` file is located in the *`install_root`*`/profiles/`*`profile_name`*`/properties/` directory of your WebSphere Application Server installation. To configure WebSphere Application Server, start the administrative console by specifying the following URL: `http://`*`server_hostname`*`:9060/ibm/console`.

Follow the documentation that accompanies your device to install your cryptographic device. Installation instructions for IBM cryptographic hardware devices can be found in the Administration section of "Security: Resources for learning" on page 879.

Note: You cannot use cryptographic token devices when you enable the Federal Information Processing Standard (FIPS) option on the Global security administrative console panel.

1. To configure a client to use a cryptographic token, edit the `sas.client.props` file and set the following properties. Fill in the **KeyStore File Name**, **KeyStore File Password**, **TrustStore File Name**, and **TrustStore File Password** fields in the Secure Sockets Layer (SSL) configuration. Leave the `com.ibm.ssl.tokenType`, `com.ibm.ssl.tokenLibraryFile`, and `com.ibm.ssl.tokenPassword` fields blank.
2. Configure your server to use the cryptographic device.

   Fill in the **KeyStore File Name**, **KeyStore File Password**, **TrustStore File Name**, **TrustStore File Password** fields in an SSL configuration. You can modify an existing configuration if you click **Security > SSL >** *alias*. You must specify an alias and select the **Cryptographic token** option. The following directions explain how to configure WebSphere Application Server for a new cryptographic device.

   a. Specify `http://`*`server_hostname`*`:9060/ibm/console` to start the administrative console.

   b. Click **Security > SSL** to open the SSL Configuration Repertoires panel. You must decide if you want to modify existing SSL repertoire entries to convert them to use hardware cryptographic devices, or create new SSL repertoire entries for the new configuration. The former is easiest as this does not require you to change any of the alias references elsewhere in the configuration.

Each protocol picks up the new configuration since it's already referencing these existing aliases. The latter is a little more difficult as you might not change every location that needs to be referenced by the new aliases. However, you have more control over which protocols actually use the cryptographic token device. If you want a specific protocol to use the cryptographic token device, it is best to create a new SSL repertoire for the cryptographic token device, then associate the alias of the new SSL repertoire with the SSL configuration of the specific protocol.

c. Click **New JSSE Repertoire** to create a new SSL setting alias if you do not want to use the default.

d. Specify an alias name in the **alias** field for the new cryptographic device. After you configure the cryptographic device, the alias appears on the Secure Sockets Layer (SSL) configuration repertoires panel. To access the panel, click **Security > SSL**.

e. Select **Cryptographic token** check box and click **OK**. This opens the **Cryptographic token - General Properties** panel.

f. Make sure the SSL configurations when associated with a transport have the appropriate signers added to the truststore or cryptographic token device so that they can contact all servers for which they are configured. For example, any CSIv2 outbound transport should have signers for all CSIv2 inbound transports that they are connecting to. This means that all CSIv2 inbound keystores (or cryptographic token devices) must have the public key of personal certificates extracted, and added as signers to the CSIv2 outbound truststores (or cryptographic token devices).

g. The following lists the locations of where SSL configuration repertoire aliases are used in the WebSphere Application Server configuration:

   For any transports that use the new NIO channel chains, including HTTP and JMS, you can modify the aliases from the following location 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)**.
   - Click **System administration > Deployment manager**. Under Additional properties, 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)**.
   - Click **System administration > Node agents >***node_agent _name*. Under Additional properties, 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 > Global security**. Under Authentication, click **Authentication protocol > CSIv2 Inbound Transport**.
   - Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 Outbound Transport**.
   - Click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS Inbound Transport**.
   - Click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS Outbound Transport**.

   For the ORB SSL transports on the server level for WebSphere Application Server Network Deployment, you can modify the SSL configuration repertoire aliases in the following locations:

   - Click **Servers > Application servers >***server_name*. Under Security, click **Server security**. Under Additional properties, click **CSIv2 Inbound Transport**.
   - Click **Servers > Application servers >***server_name*. Under Security, click **Server security**. Under Additional properties, click **CSIv2 Outbound Transport**.

- Click **Servers > Application servers >***server_name*. Under Security, click **Server security**. Under Additional properties, click **SAS Inbound Transport**.
- Click **Servers > Application servers >***server_name*. Under Security, click **Server security**. Under Additional properties, click **SAS Outbound Transport**.

For the Simple Object Access Protocol (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 **sslConfig** and select an alias in the Value field.

For additional SOAP JMX administrative transports for WebSphere Application Server Network Deployment, you can modify the SSL configuration repertoire aliases in the following locations:

- Click **System administration > Deployment manager**. Under Additional properties, click **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 **sslConfig** and select an alias in the Value field.
- Click **System administration > Node agents >***node_agent_name*. Under Additional properties, **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 **sslConfig** and select an alias in the Value field.

For the Lightweight Directory Access Protocol (LDAP) SSL transport, you can modify the SSL configuration repertoire aliases by clicking **Security > Global security**. Under User registries, click **LDAP**.

h. Finish configuring the SSL settings for this alias. When using hardware cryptographic tokens, you must use a JSSE provider of type IBMJSSE2. The IBMPKCS11Impl provider only works with the IBMJSSE2 provider.

3. Now that you have the aliases configured in the **SSL configuration repertoires** panel, you must associate the aliases with each protocol that needs to use them. If you edited existing aliases, you do not need to make any changes since they are already associated with SSL protocols. However, if you created new aliases and want to rearrange this existing alias association, then proceed to the next step.
4. Repeat steps a. through l. to edit existing or create new SSL configuration repertoires for creating a cryptographic token configuration for use by the IBMJSSE2 provider.
5. Click **OK** to complete the editing of the SSL configuration repertoire for this alias.

The WebSphere Application Server configuration is configured to take advantage of a cryptographic token device for cryptographic functions used by SSL This can improve the system performance over software encryption when SSL is used to protect your data that is transferred over the network.

WebSphere Application Server uses the cryptographic token as a keystore file for the SSL connection.

If the server configuration has changed, restart the configured server.

Using Java Secure Socket Extension and Java Cryptography Extension with Servlets and enterprise bean files

Java Secure Socket Extension

Java Secure Socket Extension (JSSE) provides the transport security for WebSphere Application Server. It provides application programming interface (API) framework and the implementation of the APIs, for Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, including functionality for data encryption, message integrity and authentication.

JSSE APIs are integrated into the Java 2 SDK, Standard Edition (J2SDK), Version 1.4. The API package for JSSE APIs is javax.net.ssl.*. Documentation for using JSSE APIs can be found in the J2SE 1.4.2 JavaDoc located at http://java.sun.com/j2se/1.4.2/docs/api/index.html.

Several JSSE providers ship with the J2SDK Version 1.4 that comes with WebSphere Application Server. The IBMJSSE provider is used in previous WebSphere releases. Associated with the IBMJSSE provider is the IBMJSSEFIPS provider, which is used when FIPS is enabled on the server. Both of these providers do not work with the JMS and HTTP transports in WebSphere Application Server Version 6. These transports take advantage of the J2SDK Verison 1.4 network input/output (NIO) asynchronous channels.

The HTTP and JMS transports use a new IBMJSSE2 provider. All other transports in WebSphere Application Server Version 6 currently use the IBMJSSE2 provider, but can be switched to the old IBMJSSE provider, if necessary (specified in the SSL repertoire configuraiton).

For more information on the new IBMJSSE2 provider, please review the documentation located in http://www.ibm.com/developerworks/java/jdk/security/142/jsse2docs.zip. After it is unzipped, the JSSE2 Reference Guide can be found at jsse2Docs/JSSE2RefGuide.html, the JSSE2 API documentation can be found at jsse2Docs/api/index.html and finally, the JSSE2 samples can be found at jsse2Docs/samples.

Customizing Java Secure Socket Extension

You can customize a number of aspects of JSSE by plugging in different implementations of Cryptography Package Provider, X509Certificate and HTTPS protocols, or specifying different default keystore files, key manager factories and trust manager factories. A provided table summarizes which aspects can be customized, what the defaults are, and which mechanisms are used to provide customization. Some of the key customizable aspects follow:

Customizable item	Default	How to customize
X509Certificate	X509Certificate implementation from IBM	cert.provider.x509v1 security property
HTTPS protocol	Implementation from IBM	java.protocol.handler.pkgs system property
Cryptography Package Provider	IBMJSSE	A security.provider.n= line in security properties file. See description.
Default keystore	None	* javax.net.ssl.keyStore system property
Default truststore	jssecacerts, if it exists. Otherwise, cacerts	* javax.net.ssl.trustStore system property
Default key manager factory	`IbmX509`	ssl.KeyManagerFactory.algorithm security property
Default trust manager factory	`IbmX509`	ssl.TrustManagerFactory.algorithm security property

For aspects that you can customize by setting a system property, statically set the system property by using the `-D` option of the Java command (you can set the system property using the administrative console), or set the system property dynamically by calling the `java.lang.System.setProperty` method in your code: `System.setProperty(propertyName,"propertyValue")`.

For aspects that you can customize by setting a Java security property, statically specify a security property value in the `java.security` properties file located in the *`install_root`*`/java/jre/lib/security` directory. The security property is `propertyName=propertyValue`. Dynamically set the Java security property by calling the `java.security.Security.setProperty` method in your code.

Application Programming Interface

The JSSE provides a standard application programming interface (API) available in packages of the `javax.net` file, `javax.net.ssl` file, and the `javax.security.cert` file. The APIs cover:

- Sockets and SSL sockets
- Factories to create the sockets and SSL sockets
- Secure socket context that acts as a factory for secure socket factories
- Key and trust manager interfaces
- Secure HTTP URL connection classes
- Public key certificate API

Samples using Java Secure Socket Extension

The Java Secure Socket Extension (JSSE) also provides samples to demonstrate its functionality. The Java Secure Socket Extension (JSSE) also provides samples to demonstrate its functionality. Download and unzip the samples included in the `http://www.ibm.com/developerworks/java/jdk/security/142/jsse2Docs.zip` file. Look in the `jsse2Docs/samples/` directory for the following files:

Files	Description
`ClientJsse.java`	Demonstrates a simple client and server interaction using JSSE. All enabled cipher suites are used.
`OldServerJsse.java`	Back-level samples
`ServerPKCS12Jsse.java`	Demonstrates a simple client and server interaction using JSSE with the PKCS12 keystore file. All enabled cipher suites are used.
`ClientPKCS12Jsse.java`	Demonstrates a simple client and server interaction using JSSE with the PKCS12 keystore file. All enabled cipher suites are used.
`UseHttps.java`	Demonstrates accessing an SSL or non-SSL Web server using the Java protocol handler of the`com.ibm.net.ssl.www.protocol` class. The URL is specified with the `http` or `https` prefix. The HTML returned from this site displays.

See more instructions in the source code. Follow these instructions before you run the samples.

Permissions for Java 2 security

You might need the following permissions to run an application with JSSE: (This is a reference list only.)
- java.util.PropertyPermission ″java.protocol.handler.pkgs″, ″write″
- java.lang.RuntimePermission ″writeFileDescriptor″
- java.lang.RuntimePermission ″readFileDescriptor″
- java.lang.RuntimePermission ″accessClassInPackage.sun.security.x509″
- java.io.FilePermission ″${user.install.root}${/}etc${/}.keystore″, ″read″
- java.io.FilePermission ″${user.install.root}${/}etc${/}.truststore″, ″read″

For the IBMJSSE provider:
- java.security.SecurityPermission ″putProviderProperty.IBMJSSE″
- java.security.SecurityPermission ″insertProvider.IBMJSSE″

For the SUNJSSE provider:
- java.security.SecurityPermission ″putProviderProperty.SunJSSE″
- java.security.SecurityPermission ″insertProvider.SunJSSE″

Debugging

By configuring through the `javax.net.debug` system property, JSSE provides the following dynamic debug tracing: `-Djavax.net.debug=true`.

A value of **true** turns on the trace facility. Use the administrative console to set the system property for debugging the application server.

Documentation

See the “Security: Resources for learning” on page 879 article for documentation references to JSSE.

JCE

Java Cryptography Extension (JCE) provides cryptographic, key and hash algorithms for WebSphere Application Server. It provides a framework and implementations for encryption, key generation, key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block and stream ciphers.

IBMJCE

The IBM version of the Java Cryptography Extension (IBMJCE) is an implementation of the JCE cryptographic service provider that is used in WebSphere Application Server. The IBMJCE is similar to SunJCE, except that the IBMJCE offers more algorithms:

- Cipher algorithm (AES, DES, TripleDES, PBEs, Blowfish, and so on)
- Signature algorithm (SHA1withRSA, MD5withRSA, SHA1withDSA)
- Message digest algorithm (MD5, MD2, SHA1, SHA-256, SHA-384, SHA-512)
- Message authentication code (HmacSHA1, HmacMD5)
- Key agreement algorithm (DiffieHellman)
- Random number generation algorithm (IBMSecureRandom, SHA1PRNG)
- Key store (JKS, JCEKS, PKCS12)

The IBMJCE belongs to the com.ibm.crypto.provider.* packages.

For further information, see the `http://www.ibm.com/developerworks/java/jdk/security/142/jceDocs.zip` file.

IBMJCEFIPS

The IBM version of the Java Cryptography Extension Federal Information Processing Standard (IBMJCEFIPS) is an implementation of the JCE cryptographic service provider that is used in WebSphere Application Server. The IBMJCEFIPS service provider implements the following:

- Signature algorithms (SHA1withDSA, SHA1withRSA)
- Cipher algorithms (AES, TripleDES, RSA)
- Key agreement algorithm (DiffieHellman)
- Key (pair) generator (DSA, AES, TripleDES, HmacSHA1, RSA, DiffieHellman)
- Message authentication code (MAC) (HmacSHA1)
- Message digest (MD5, SHA-1, SHA-256, SHA-384, SHA-512)
- Algorithm parameter generator (DiffieHellman, DSA)
- Algorithm parameter (AES, DiffieHellman, DES, TripleDES, DSA)
- Key factory (DiffieHellman, DSA, RSA)
- Secret key factory (AES, TripleDES)
- Certificate (X.509)

- Secure random (IBMSecureRandom)

Application Programming Interface

Java Cryptography Extension (JCE) has a provider-based architecture. Providers can be plugged into the JCE framework by implementing the APIs defined by the JCE. The JCE APIs covers:
- Symmetric bulk encryption, such as DES, RC2, and IDEA
- Symmetric stream encryption, such as RC4
- Asymmetric encryption, such as RSA
- Password-based encryption (PBE)
- Key Agreement
- Message Authentication Codes

Samples using Java Cryptography Extension

There are samples located in `http://www.ibm.com/developerworks/java/jdk/security/142/jceDocs.zip` file. Unzip the file and locate the following samples in the`jceDocs/samples` directory:

File	Description
`SampleDSASignature.java`	Demonstrates how to generate a pair of DSA keys (a public key and a private key) and use the key to digitally sign a message using the SHA1with DSA algorithm
`SampleMarsCrypto.java`	Demonstrates how to generate a Mars secret key, and how to do Mars encryption and decryption
`SampleMessageDigests.java`	Demonstrates how to use the message digest for MD2 and MD5 algorithms
`SampleRSACrypto.java`	Demonstrates how to generate an RSA key pair, and how to do RSA encryption and decryption
`SampleRSASignatures.java`	Demonstrates how to generate a pair of RSA keys (a public key and a private key) and use the key to digitally sign a message using the SHA1withRSA algorithm
`SampleX509Verification.java`	Demonstrates how to verify X509 Certificates

Documentation

Refer to the “Security: Resources for learning” on page 879 for documentation on JCE.

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 global security includes J2EE role-based authorization, the Common Secure Interoperability Version 2 (CSIv2) authentication protocol, and Secure Sockets Layer (SSL) configuration.

Since 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 should 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 in WebSphere Application Server Version 5, it is disabled by default. However, it is enabled automatically if you also enable global security when configuring security. Although it becomes enabled automatically when you enable WebSphere global security, you can choose to disable it. You can configure Java 2 security and global security independently of one another. Disabling global 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 `SystemOut.log` file, `SystemError.log` file, or the trace log 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.

There are implications 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 recommended in the J2EE 1.3 Specification. For details of default permissions granted to applications in the product, refer to the following policy files:

- *install_root*/java/jre/lib/security/java.policy
- *install_root*/properties/server.policy
- *install_root*/config/cells/*cell_name*/nodes/*node_name*/app.policy

Note: This 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 is generated from within WebSphere Application Server. The product does not support a more restrictive policy than the default defined in the policy files previously mentioned.

There are several policy files used to define the security policy for the Java process. These policy files are static (code base is defined in the policy file) and they are 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 run time. Refer to the article, Java 2 security policy files for more information.

Note: Syntax errors in the policy files cause the application server process to fail. Edit these policy files carefully using the Policy Tool provided by the IBM Developer Kit, Java Technology Edition for editing the policy files (*install_root*/java/jre/bin/policytool).

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 run time. 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 for information and strategies for dealing with such applications.

It is important to note that when Java 2 Security is enabled in the Global Security settings, the installed SecurityManager does not currently check modifyThread and modifyThreadGroup permissions for non-system threads. Allowing Web and 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 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 static policy files, such as `java.policy`, the `was.policy` (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 should document the required Java 2 permissions needed 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 should never be granted access to resources protected by the permission, then it might be necessary to mark the code as **privileged** (inserting doPrivileged). Refer to the article, AccessControlException, for more details. However, improperly inserting a doPrivileged might open up security holes. Understand the implication of doPrivileged to make a correct judgement whether a doPrivileged should be inserted or not.

The section on Dynamic Policy describes how the permissions in the `was.policy` files are granted at run time.

Developing an application with Java 2 security in mind 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/1.3/docs/guide/security/index.html`.

Debugging Aids

There are two primary aids, the WebSphere `SystemOut.log` file and the com.ibm.websphere.java2secman.norethrow property.

The WebSphere SystemOut.log File

The AccessControl exception logged in the `SystemOut.log` file 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 throws 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 `SystemOut.log` file.

However, when the JVM com.ibm.websphere.java2secman.norethrow property is set and has a value of **true**, the security manager does not throw 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**. Under Additional Properties, click **Process Definition > Java Virtual Machine > Custom Properties > New**. In the Name field, type **com.ibm.websphere.java2secman.norethrow**. In the Value field, type **true**.

To set the `com.ibm.websphere.java2secman.norethrow` property for the node agent, go to the WebSphere Application Server administrative console and click **System Administration > Node Agents**. Under Additional Properties, click **Process Definition > Java Virtual Machine > Custom Properties > New**. In the Name field, type **com.ibm.websphere.java2secman.norethrow**. In the Value field, type **true**.

To set the `com.ibm.websphere.java2secman.norethrow` property for the deployment manager, go to the WebSphere Application Server administrative console and click **System Administration > Deployment Manager**. Under Additional Properties, click **Process Definition > Java Virtual Machine > Custom Properties > New**. In the Name field, type **com.ibm.websphere.java2secman.norethrow**. In the Value field, type **true**.

Note: This property is intended for a sandbox or debug environment because it instructs the security manager not to throw the AccessControl exception. Java 2 security is not enforced. This property should not be used 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 the where the `SystemOut.log` file can be inspected for AccessControl exceptions. Since this property does not throw 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 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, there is no easy way 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 embedded in the application's .ear file, for example:

```
grant codeBase "file:${application}" {
            permission java.security.AllPermission;
        };
```

install_root/**properties/server.policy**

This policy defines the policy for the WebSphere 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 desired policy file as the value of the Java system properties java.security.policy. 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 was.policy file (per enterprise application) to define Java 2 security policy for enterprise application resources.

WAS_HOME/**java/jre/lib/security/java.policy**

The file represents the default permissions granted to all classes. The policy of this file applies to all the processes launched by the WebSphere Application Server JVM.

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() reported a SecurityException 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 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. Once 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.

Note: If the application is running with Java Mail, this message might be benign. User 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.
Recommended response:	None.

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.
Recommended response:	Contact an IBM representative.

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;
};
```

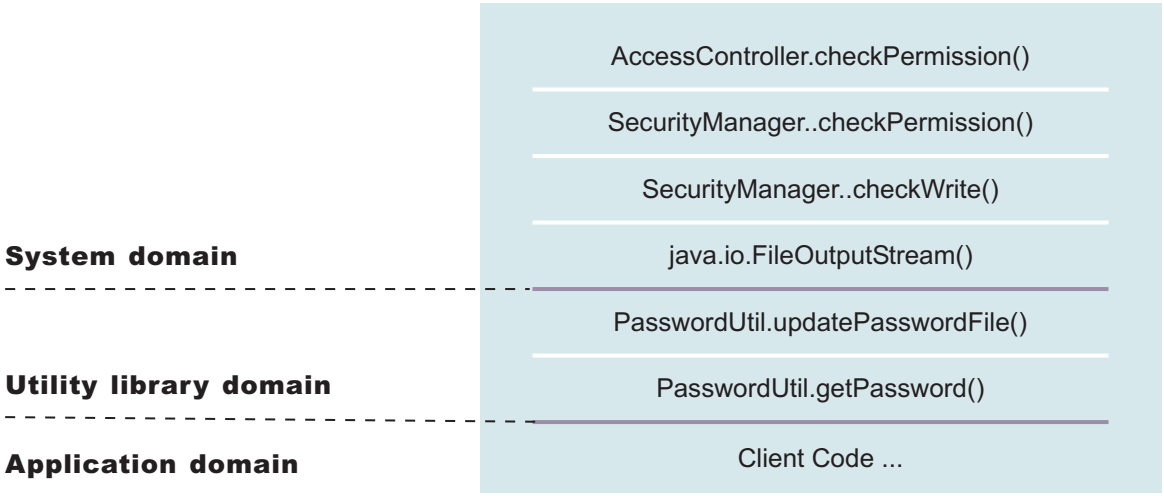
The algorithm requires that all the classes or callers on the call stack have the permissions when a java.security.AccessController.checkPermission method is performed or the request is denied and a java.security.AccessControlException exception is created. However, if the caller is marked as *privileged* 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.

Configuring Java 2 security

Java 2 security is a programming model that is very pervasive and has a huge impact on application development. It is disabled by default, but is enabled automatically when global security is enabled.

However, Java 2 security is orthogonal to Java 2 Platform, Enterprise Edition (J2EE) role-based security; you can disable or enable it independently of Global Security.

However, it does provide an extra level of access control protection on top of the J2EE role-based authorization. It particularly addresses the protection of system resources and application programming interfaces (API). Administrators should need to consider the benefits against the risks of disabling Java 2 Security.

The following recommendations are provided to help enable Java 2 security in a test or production environment:

1. Make sure the application is developed with the Java 2 security programming model in mind. Developers have to know whether or not the APIs used in the applications are protected by Java 2 security. It is very important that the required permissions for the APIs used are declared in the policy file (*was.policy*), or the application fails to run when Java 2 security is enabled. Developers can reference the Web site for Development Kit APIs that are protected by Java 2 security. See the Programming model and decisions section of the “Security: Resources for learning” on page 879 article to visit this Web site.
2. Make sure that migrated applications from previous releases are given the required permissions. Since Java 2 security is not supported or partially supported in previous WebSphere Application Server releases, applications developed prior to Version 5 most likely are not using the Java 2 security programming model. There is no easy way to find out all the required permissions for the application. Following are activities you can perform to determine the extra permissions required by an application:
   - Code review and code inspection
   - Application documentation review
   - Sandbox testing of migrated enterprise applications with Java 2 security enabled in a pre-production environment. Enable tracing in WebSphere Java 2 security manager to help determine the missing permissions in the application policy file. The trace specification is `com.ibm.ws.security.core.SecurityManager=all=enabled`.
   - Use the `com.ibm.websphere.java2secman.norethrow` system property to aid debuggging. This property should not be used in a production environment. Refer to “Java 2 security” on page 1208.

Note: The default permission set for applications is the recommended permission set defined in the J2EE 1.3 Specification. The default is declared in the `profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/nodes/`*`node_name`*`/app.policy` policy file with permissions defined in the Development Kit (*`JAVA_HOME`*`/jre/lib/security/java.policy`) policy file that grant permissions to everyone. However, applications are denied permissions declared in the `profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/filter.policy` filter policy file. Permissions declared in the *filter.policy* file are filtered for applications during the permission check.

Note: Define the required permissions for an application in a `was.policy` file and embed the `was.policy` file in the application enterprise archive (EAR) file as *YOURAPP.ear/META-INF/was.policy* (see “Configuring Java 2 security policy files” on page 1222 for details).

The following steps describe how to enforce Java 2 security on the cell level for WebSphere Application Server Network Deployment and the server level for WebSphere Application Server and WebSphere Application Server Express:

1. Click **Security > Global security**. The Global security panel is displayed.
2. Select the **Enforce Java 2 security** option.
3. Click **OK** or **Apply**.
4. Click **Save** to save the changes.
5. Restart the server for the changes to take effect.

Java 2 security is enabled and enforced for the servers. Java 2 security permission is selected when a Java 2 security protected API is called.

When to use Java 2 security

1. To enable protection on system resources. For example, when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java Virtual Machine system properties, and so on.
2. To prevent application code calling destructive APIs. For example, calling the *System.exit()* method brings down the application server.
3. To prevent application code from obtaining privileged information (passwords) or gaining extra privileges (obtaining server credentials).

You can enforce Java 2 security on the server level for WebSphere Application Server Network Deployment completing the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Server security**.
3. Under Additional properties, click **Server-level security**.
4. Select the **Enforce Java 2 security** option.
5. Click **OK** or **Apply**.
6. Click **Save** to save the changes.
7. Restart the server for the changes to take effect.

The WebSphere Java 2 security manager is enhanced to dump the Java 2 security permissions granted to all classes on the call stack when an application is denied access to a resource (the `java.security.AccessControlException` exception is thrown). However, this tracing capability is disabled by default. You can enable it by specifying the server trace service with the `com.ibm.ws.security.core.SecurityManager=all=enabled` trace specification. When the exception is thrown, the trace dump provides hints to determine whether the application is missing permissions or the product run time code or third party libraries used are not properly marked as *privileged* when accessing Java 2 protected resources. See the Security Problem Determination Guide for details.

Using PolicyTool to edit policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See Dynamic policy for the list of available policy files. The Java Development Kit provides *policytool* to edit these policy files. This tool is recommended for editing any policy file to verify the syntax of its contents. Syntax errors in the policy file cause an *AccessControlException* during application execution, including the server start. Identifying the cause of this exception is not easy because the user might not be familiar with the resource that has an access violation. Be careful when you edit these policy files.

To use the policytool with WebSphere Application Server for z/OS, choose one of the following two options:

- Move the policy files to another platform such as Microsoft Windows and modify the files. To use this option, you must issue the FTP command to transfer the files to the other platform, invoke the policytool, and transfer the updated files back to z/OS in binary mode.
- Invoke the policytool on z/OS that is supplied with the Software Development Kit (SDK) installed on your z/OS system. For more information on this option, complete the following steps:
  1. Export the display to an Xwindows-enabled device. For example, in Open MVS (OMVS), type `export DISPLAY=<IP_address_of_the_Xwindows_device>:0.0`
  2. Enable the z/OS system to access the display of the Xwindows-enabled device. For example, on AIX, type `xhost + <address_of_the_MVS_system>`.
  3. Convert the policy file to the Extended Binary Coded Decimal Interchange Code (EBCDIC) format.
  4. Invoke the policytool on OMVS by typing `<$JAVA_HOME>/policytool`. <$*JAVA_HOME*> is the directory in which the SDK is installed.
  5. Click **File** > **Open**.

6. Navigate the directory tree in the **Open** window to pick up the policy file that you need to update. After selecting the policy file, click **Open**. The code base entries are listed in the window.
7. Create or modify the code base entry.
   a. Modify the existing code base entry by double-clicking the code base, or click the code base and click **Edit Policy Entry**. The Policy Entry window opens with the permission list defined for the selected code base.
   b. Create a new code base entry by clicking **Add Policy Entry**. The Policy Entry window opens. At the code base column, enter the code base information as a URL format, for example, `/WebSphere/AppServer/InstalledApps/testcase.ear`.
8. Modify or add the permission specification
   a. Modify the permission specification by double-clicking the entry you want to modify, or by selecting the permission and clicking **Edit Permission**. The Permissions window opens with the selected permission information.
   b. Add a new permission by clicking **Add Permission**. The Permissions window opens. In the Permissions, window there are four rows for **Permission**, **Target Name**, **Actions**, and **Signed By**.
9. Select the permission from the Permission list. The selected permission displays. After a permission is selected, the **Target Name**, **Actions**, and **Signed By** fields automatically show the valid choices or they enable text input in the right text input area.
   a. Select **Target Name** from the list, or enter the target name in the right text input area.
   b. Select **Actions** from the list.
   c. Input **Signed By** if it is needed.

   Important: The Signed By keyword is not supported in the following policy files: `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the Signed By keyword is supported in the following policy files:`java.policy`, `server.policy`, and `client.policy` files. The Java Authentication and Authorization Service (JAAS) is not supported in the `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`.
10. Click **OK** to close the Permissions window. Modified permission entries of the specified code base display.
11. Click **Done** to close the window. Modified code base entries are listed. Repeat steps 4 through 8 until you complete editing.
12. Click **File** > **Save** after you finish editing the file.
13. Convert the policy file back from the EBCDIC format to the ASCII format.

A policy file is updated. If any policy files need editing, use the policytool. Do not edit the policy file manually. Syntax errors in the policy files can potentially cause application servers or enterprise applications to not start or function incorrectly. For the changes in the updated policy file to take effect, restart the Java processes.

Java 2 security policy files:

The Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 specification has 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	*install_root*/java/jre/lib/security/java.policy. Default permissions granted to all classes. The policy of this file applies to all the processes launched by WebSphere Application Server.
server.policy	*install_root*/profiles/*profile_name*/properties/server.policy. Default permissions granted to all the product servers.
client.policy	*install_root*/profiles/*profile_name*/properties/client.policy. Default permissions 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	*install_root*/profiles/*profile_name*/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 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 specified in the class path of a ResourceAdapter. The default permission of the spi.policy file is java.security.AllPermissions.
library.policy	*install_root*/profiles/*profile_name*/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 is empty.
app.policy	*install_root*/profiles/*profile_name*/config/cells/*cell_name* /nodes/*node_name*/app.policy The app.policy file defines the default permissions that are granted to all the enterprise applications running on *node_name* in *cell_name*.
was.policy	*install_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.

Note: 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:${application}	Permissions apply to all resources within the application
file:${jars}	Permissions apply to all utility Java archive (JAR) files within the application
file:${ejbComponent}	Permissions apply to Enterprise JavaBeans (EJB) resources within the application
file:${webComponent}	Permissions apply to Web resources within the application
file:${connectorComponent}	Permissions apply to connector resources within the application

Other than these entries specified by the code base symbols, you can specify the module name for a granular setting. 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:${application}	Permissions apply to all resources within the application
file:${jars}	Permissions apply to all utility JAR files within the application
file:${ejbComponent}	Permissions apply to enterprise beans resources within the application
file:${webComponent}	Permissions apply to Web resources within the application
file:${connectorComponent}	Permissions apply to connector resources both within the application and in stand-alone 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

Note: 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 run time, 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 the keyword, filterMask, the run time 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 keyword, runtimeFilterMask, the run time 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 runtimeFilterMask are removed from the granted permission during run time.

If the Issue Permission Warning flag in the Global Security 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 (*install_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 1.4.2. 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 loaded by the IBM Developer Kit, Java Technology Edition Version 1.4.2 policy tool, 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 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.

Configuring Java 2 security policy files:

Java 2 security uses several policy files to determine the granted permissions for each Java programs. See the “Java 2 security policy files” on page 1217 article for the list of available policy files supported by WebSphere Application Server.

There are two types of policy files supported by WebSphere Application Server: dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application permissions. There are six dynamic policy files:

Policy file name	Description
`app.policy`	Contains default permissions for all of the enterprise applications in the cell.
`was.policy`	Contains application-specific permissions for an WebSphere Application Server enterprise application. This file is packaged in an enterprise archive (EAR) file.
`ra.xml`	Contains connector application specific permissions for a WebSphere Application Server enterprise application. This file is packaged in a resource adapter archive (RAR) file.
`spi.policy`	Contains permissions for Service Provider Interface (SPI) or third-party resources embedded in WebSphere Application Server. The default contents grant everything. Update this file carefully when the cell requires more protection against SPI in the cell. This file is applied to all of the SPIs defined in the `resources.xml` file.
`library.policy`	Contains permissions for the shared library of enterprise applications.
`filter.policy`	Contains the list of permissions that require filtering from the `was.policy` file and the `app.policy` file in the cell. This filtering mechanism only applies to the `was.policy` and `app.policy` files.

In WebSphere Application Server, applications must have the appropriate thread permissions specified in the was.policy or app.policy file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server throws a java.security.AccessControlException. The app.policy file applies to a specified node. If you change the permissions in one app.policy file, you must incorporate the new thread policy in the same file on the remaining nodes. Also, if you add the thread permissions to the app.policy file, you must restart WebSphere Application Server to enforce the new permissions. However, if you add the permissions to the was.policy file for a specific application, you do not need to restart WebSphere Application Server. An administrator must add the following code to a was.policy or app.policy file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
permission java.lang.RuntimePermission "stopThread";
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

Important: The Signed By keyword is not supported in the following policy files: app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the Signed By keyword is supported in the following policy files:java.policy, server.policy, and client.policy files. The Java Authentication and Authorization Service (JAAS) is not supported in the app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=*URL* where *URL* is the location of the authorization policy.

1. Identify the policy file to update.
   - If the permission is required by an application, update the static policy file. Refer to “Configuring static policy files” on page 1235.
   - If the permission is required by all of the WebSphere Application Server enterprise applications in the node, refer to “Configuring spi.policy files” on page 1231.
   - If the permission is required only by specific WebSphere Application Server enterprise applications and the permission is required only by connector, update the ra.xml file. Refer to Assembling resource adapter (connector) modules. Otherwise, update the was.policy file. Refer to “Configuring the was.policy file” on page 1228 and “Adding the was.policy file to applications” on page 1233.
   - If the permission is required by shared libraries, refer to “Configuring library.policy files” on page 1232.
   - If the permission is required by SPI libraries, refer to “Configuring spi.policy files” on page 1231.

   Note: It is recommended to pick up the policy file with the smallest scope. You can avoid giving an extra permission to the Java programs and protect the resources. You can update the ra.xml file or the was.policy file rather than the app.policy file. Use specific component symbols ($(ejbcomponent), ${webComponent},${connectorComponent} and ${jars}) than ${application} symbols. Update dynamic policy files than static policy files.

   Add any permission that should never be granted to the WebSphere Application Server enterprise application in the cell to the filter.policy file. Refer to “Configuring filter.policy files” on page 1226.
2. Restart the WebSphere Application Server enterprise application.

The required permission is granted for the specified WebSphere Application Server enterprise application.

If an WebSphere Application Server enterprise application in a cell requires permissions, some of the dynamic policy files need updating. The symptom of the missing permission is the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines were split onto two lines because of the width of the page. However, the permission should be on one line.

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate dynamic policy file, for example,

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read";
};
```

The previous two lines were split onto two lines because of the width of the page. However, the permission should be on one line.

To decide whether to add a permission, refer to the article “Access control exception” on page 1213.

Configuring app.policy files:

Java 2 security uses several policy files to determine the granted permissions for each Java program. See the Dynamic policy article for the list of available policy files supported by WebSphere Application Server. The `app.policy` file is a default policy file shared by all of the WebSphere Application Server enterprise applications. The union of the permissions contained in the following files is applied to the WebSphere Application Server enterprise application:

- Any policy file that is specified in the policy.url.* properties in the `java.security` file.
- The `app.policy` files, which are managed by configuration and file replication services.
- The `server.policy` file.
- The `java.policy` file.
- The application `was.policy` file.
- The permission specification of the `ra.xml` file.
- The shared library, which is the `library.policy` file.

Changes made in these files are replicated to other nodes in the Network Deployment cell.

In WebSphere Application Server, applications that manipulate threads must have the appropriate thread permissions specified in the `was.policy` or `app.policy` file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server throws a java.security.AccessControlException. If an administrator adds thread permissions to the app.policy file, the permission change requires a restart of the WebSphere Application Server. An administrator must add the following code to a `was.policy` or `app.policy` file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
permission java.lang.RuntimePermission "stopThread";
permission java.lang.RuntimePermission "modifyThread";
 permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

Important: The Signed By and the Java Authentication and Authorization Service (JAAS) principal keywords are not supported in the `app.policy` file. However, the Signed By keyword is supported in the following files: `java.policy`, `server.policy`, and the `client.policy` files. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java

Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=*URL* where *URL* is the location of the authorization policy.

If the default permissions for enterprise applications (the union of the permissions defined in the java.policy file, the server.policy file and the app.policy file) are enough, no action is required. The default app.policy file is used automatically. If a specific change is required to all of the enterprise applications in the cell, update the app.policy file. Syntax errors in the policy files cause start failures in the application servers. Edit these policy files carefully.

1. Extract the policy file. Enter the following command on one line at a command prompt:

   ```
   wsadmin  wsadmin> set obj [$AdminConfig extract profiles/profile_name/cells/cell_name
   /node/node_name/app.policy c:/temp/test/app.policy].
   ```

2. Edit the extracted app.policy file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 1216.
3. Check in the policy file. Enter the following command on one line at a command prompt:

   ```
   wsadmin> $AdminConfig checkin profile/profile_name/cells/cell_name
   /nodes/node_name/app.policy c:/temp/test/was.policy $obj.
   ```

The default Java 2 security policies have been changed for the enterprise application.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resource.

Symbol	Meaning
file:${application}	Permissions apply to all resources within the application
file:${jars}	Permissions apply to all utility Java archive (JAR) files within the application
file:${ejbComponent}	Permissions apply to enterprise bean resources within the application
file:${webComponent}	Permissions apply to Web resources within the application
file:${connectorComponent}	Permissions apply to connector resources both within the application and within stand-alone connector resources.

There are five embedded symbols provided to specify the path and name for java.io.FilePermission. These symbols enable flexible permission specifications. 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

Note: You cannot use the ${was.module.path} in the ${application} entry.

The app.policy file supplied by WebSphere Application Server resides at *install_root*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/app.policy, which contains the following default permissions:

Attention: In the following code sample, the first two lines related to `permission java.io.FilePermission` were split into two lines each due to the width of the printed page.

```
grant codeBase "file:${application}" {
  // The following are required by Java mail
  permission java.io.FilePermission "${was.install.root}${/}java${/}
  jre${/}lib${/}ext${/}mail.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}java${/}
  jre${/}lib${/}ext${/}activation.jar", "read";
};

grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${ejbComponent}" {
 permission java.lang.RuntimePermission "queuePrintJob";
 permission java.net.SocketPermission "*", "connect";
 permission java.util.PropertyPermission "*", "read";
};
```

If all of the WebSphere Application Server enterprise applications in a cell require permissions that are not defined as defaults in the `java.policy` file, the `server.policy` file and the `app.policy` file, then update the `app.policy` file. The symptom of a missing permission is the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When a Java program receives this exception and adding this permission is justified, add a permission to the `server.policy` file, for example:

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to the article AccessControlException.

Restart all WebSphere Application Server enterprise applications to ensure that the updated `app.policy` file takes effect.

Configuring filter.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. Java 2 security policy filtering is only in effect when Java 2 security is enabled. Refer to Configuring Java 2 security. The filtering policy defined in the `filter.policy` file is cell wide. Refer to the article, Dynamic policy, for the list of available policy files supported by WebSphere Application Server. The `filter.policy` file is the only policy file used when restricting the permission instead of granting permission. The permissions listed in the filter policy file are filtered out from the `app.policy` file and the `was.policy` file. Permissions defined in the other policy files are not affected by the `filter.policy` file.

When a permission is filtered out, an audit message is logged. However, if the permissions defined in the `app.policy` file and the `was.policy` file are compound permissions like java.security.AllPermission, for example, the permission is not removed. A warning message is logged. If the Issue Permission Warning flag is enabled (default) and if the `app.policy` file and the `was.policy` file contain custom permissions (non-Java API permission, the permission package name begins with characters other than `java` or `javax`), then a warning message is logged and the permission is not removed. You can change the value of the **Issue permission warning** option on the Global Security panel. It is not recommended that you use `AllPermission` for the enterprise application.

There are some default permissions defined in the `filter.policy` file. These permissions are the minimal ones recommended by the product. If more permissions are added to the `filter.policy` file, certain operations can fail for enterprise applications. Add permissions to the `filter.policy` file carefully.

1. Extract the `filter.policy` file.
   a. From the command prompt, enter `wsadmin wsadmin> set obj [$AdminConfig extract profiles/`*`profile_name`*`/cells/`*`cell_name`*`/filter.policy c:/temp/test/filter.policy]`
2. You cannot use the Policy Tool to edit the `filter.policy` file. Editing must be completed in a text editor. Be careful and verify that there are no syntax errors in the `filter.policy` file. If there are any syntax errors in `filter.policy` file, it will not be loaded by the product security run time, which implies that filtering is disabled.
3. Check in the policy file.
   a. Type the following from a command prompt: `wsadmin> $AdminConfig checkin profiles/`*`profile_name`*`/cells/`*`cell_name`*`/filter.policy c:/temp/test/filter.policy $obj.`

An updated `filter.policy` file is applied to all of the WebSphere Application Server enterprise application after the servers are restarted.

The `filter.policy` file is managed by configuration and file replication services. Changes made in the file are replicated to other nodes in the Network Deployment cell.

The `filter.policy` file supplied by WebSphere Application Server resides at:
`install_root``/profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/filter.policy`.

It contains these permissions as defaults:

```
filterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
runtimeFilterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
```

The permissions defined in `filterMask` are for static policy filtering. The security run time tries to remove the permissions from applications during application startup. Compound permissions are not removed but

are issued with a warning, and application deployment is stopped if applications contain permissions defined in `filterMask`, and if scripting was used (wsadmin tool). The `runtimeFilterMask` defines permissions used by the security run time to deny access to those permissions to application thread. Do not add more permissions to the `runtimeFilterMask`. Application start failure or incorrect functioning might result. Be careful when adding more permissions to the `runtimeFilterMask`. Usually, you only need to add permissions to the `filterMask` stanza.

WebSphere Application Server relies on the filter policy file to restrict or disallow certain permissions that could compromise the integrity of the system. For instance, WebSphere Application Server considers the `exitVM` and `setSecurityManager` permissions as those permissions that most applications should never have. If these permissions are granted, then the following scenarios are possible:

exitVM
A servlet, JSP file, enterprise bean, or other library used by the aforementioned might call the System.exit() API and cause the entire WebSphere Application Server process to terminate.

setSecurityManager
An application might install its own security manager and either grant more permissions or bypass the default policy that the WebSphere Application Server security manager enforces.

Important: 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 **Enforce Java 2 Security** option on the Global security settings 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).

For the updated `filter.policy` file to take effect, restart related Java processes.

Configuring the was.policy file:

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 1217 for the list of available policy files supported by WebSphere Application Server Version 6.0.x. The `was.policy` file is an application-specific policy file for WebSphere Application Server enterprise applications. It is embedded in the enterprise archive (EAR) file (`META-INF/was.policy`). The `was.policy` file is located in:

install_root/profiles/*profile_name*/config/cells/*cell_name*/applications/
ear_file_name/deployments/*application_name*/META-INF/was.policy

The union of the permissions contained in the following files is applied to the WebSphere Application Server enterprise application:

- Any policy file that is specified in the policy.url.* properties in the `java.security` file.
- The `app.policy` files, which are managed by configuration and file replication services.
- The `server.policy` file.
- The `java.policy` file.
- The application `was.policy` file.
- The permission specification of the `ra.xml` file.
- The shared library, which is the `library.policy` file.

Changes made in these files are replicated to other nodes in the Network Deployment cell.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

Symbol	Definition
file:${application}	file:${application}
file:${jars}	Permissions apply to all utility Java archive (JAR) files within the application
file:${ejbComponent}	Permissions apply to enterprise bean resources within the application
file:${webComponent}	Permissions apply to Web resources within the application
file:${connectorComponent}	Permissions apply to connector resources within the application

In WebSphere Application Server, applications that manipulate threads must have the appropriate thread permissions specified in the `was.policy` or `app.policy` file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server throws a java.security.AccessControlException. If you add the permissions to the `was.policy` file for a specific application, you do not need to restart WebSphere Application Server. An administrator must add the following code to a `was.policy` or `app.policy` file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
permission java.lang.RuntimePermission "stopThread";
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

An administrator can add the thread permissions to the app.policy file, but the permission change requires a restart of the WebSphere Application Server.

Important: The Signed By and the Java Authentication and Authorization Service (JAAS) principal keywords are not supported in the `was.policy` file. The **Signed By** keyword is supported in the following policy files: `java.policy`, `server.policy`, and `client.policy`. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`. You can statically set the authorization policy files in `java.security.auth.policy` with `auth.policy.url.n=`*`URL`* where *`URL`* is the location of the authorization policy.

Other than these blocks, you can specify the module name for granular settings. For example,

```
"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";
};
```

There are five embedded symbols provided to specify the path and name for the `java.io.FilePermission`. These symbols enable flexible permission specification. The absolute file path is fixed after the application is installed.

Symbol	Definition
${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

If the default permissions for the enterprise application (union of the permissions defined in the `java.policy` file, the `server.policy` file and the `app.policy` file) are enough, no action is required. If an application has specific resources to access, update the `was.policy` file. The first two steps assume that you are creating a new policy file.

Note: Syntax errors in the policy files cause the application server to fail. Use care when editing these policy files.

1. Create or edit a new `was.policy` file using the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 1216.
2. Package the `was.policy` file into the enterprise archive (EAR) file.

   For more information, see “Adding the was.policy file to applications” on page 1233.The following instructions describe how to import a `was.policy` file.

   a. Import the EAR file into an assembly tool. For more information, see Importing enterprise applications.
   b. Open the Project Navigator view.
   c. Expand the EAR file and click **META-INF**. You might find a `was.policy` file in the META-INF directory. If you want to delete the file, right-click the file name and select **Delete**.
   d. At the bottom of the Project Navigator view, click **J2EE Hierarchy**.
   e. Import the `was.policy` file by right-clicking the **Modules** directory within the deployment descriptor and clicking **Import > Import > File system**.
   f. Click **Next**.
   g. Enter the path name to the `was.policy` file in the **From directory** field or click **Browse** to locate the file.
   h. Verify that the path directory listed in the **Into directory** field lists the correct `META-INF` directory.
   i. Click **Finish**.
   j. To validate the EAR file, right-click the EAR file, which contains the Modules directory, and click **Run Validation**.
   k. To save the new EAR file, right-click the EAR file, and click **Export > Export EAR file**. If you do not save the revised EAR file, the EAR file will contain the new `was.policy` file. However, if the workspace becomes corrupted, you might lose the revised EAR file.
   l. To generate deployment code, right-click the EAR file and click **Generate Deployment Code**.
3. Update an existing installed application, if one already exists.
   a. Modify the was.policy file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 1216.
   b. Extract the policy file. Enter the following from a command prompt:

      ```
      wsadmin  wsadmin>  set obj [$AdminConfig extract profiles/profile_name/cells/cell_name
      /application/ear_file_name/deployments/application_name
      /META_INF/was.policy c:/temp/test/was.policy]
      ```

      Enter the three previous lines as one continuous line.

c. Edit the extracted was.policy file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 1216.

d. Check in the policy file. Enter the following at a command prompt:

```
wsadmin> $AdminConfig checkin profiles/profile_name/cells/cell_name/application/
ear_file_name/deployments/application_name/META_INF/was.policy
c:/temp/test/was.policy $obj
```

Enter the three previous lines as one continuous line.

The updated was.policy file is applied to the application after the application restarts.

If an application must access a specific resource that is not defined as a default in the java.policy file, the server.policy file and the app.policy, then delete the was.policy file for that application. The symptom of the missing permission is that the exception, java.security.AccessControlException. The missing permission is listed in the exception data, java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read).

When a Java program receives this exception and adding this permission is justified, add a permission to the was.policy file: grant codeBase ″file:<user client installed location>″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar″, ″read″; };.

To determine whether to add a permission, refer to the article, “Access control exception” on page 1213.

Restart all applications for the updated app.policy file to take effect.

Configuring spi.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 1217 for the list of available policy files supported by WebSphere Application Server Version 6.0.x.

Since the default permissions for Service Provider Interface (SPI) is AllPermission, the only reason to update the spi.policy file is a restricted SPI permission. When a change in the spi.policy is required, complete the following steps.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

Important: Do not place the codebase keyword or any other keyword after the filterMask and runtimeFilterMask keywords. The Signed By and the Java Authentication and Authorization Service (JAAS) Principal keywords are not supported in the spi.policy file. The Signed By keyword is supported in the following policy files: java.policy, server.policy, and client.policy. The JAAS Principal keyword is supported in a JAAS policy file that is specified by the Java Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=*URL* where *URL* is the location of the authorization policy.

1. Extract the policy file.
   a. From the command prompt, enter wsadmin> set obj [$AdminConfig extract profiles/*profile_name*/cells/*cell_name*/nodes/*node_name*/spi.policy c:/temp/test/spi.policy]
2. Edit the extracted spi.policy with the Policy Tool.
3. Check in the policy file.
   a. Enter the following from a command prompt wsadmin> $AdminConfig checkin profiles/*profile_name*/cells/*cell_name*/nodes/*node_name*/spi.policy c:/temp/test/spi.policy $obj.

The updated spi.policy is applied to the SPI libraries after the Java process is restarted.

The spi.policy file is the template for SPIs (Service Provider Interface) or third-party resources embedded in the product. Example of SPIs are Java Message Services (JMS) (MQSeries) and Java database connectivity (JDBC) drivers. They are specified in the resources.xml file. The dynamic policy grants the permissions defined in the spi.policy file to the class paths defined in the resources.xml file. The union of the permission contained in the java.policy file and the spi.policy file are applied to the SPI libraries. The spi.policy files are managed by configuration and file replication services. Changes made in these files are replicated to other nodes in the Network Deployment cell.

The spi.policy file supplied by WebSphere Application Server resides at *install_root*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/spi.policy. It contains the following default permission:

```
grant {
   permission java.security.AllPermission;
};
```

Restart the related Java processes for the changes in the spi.policy file to become effective.

Configuring library.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See "Java 2 security policy files" on page 1217 for the list of available policy files supported by WebSphere Application Server. The library.policy file is the template for shared libraries (Java library classes). Multiple enterprise applications can define and use shared libraries. Refer to Managing shared libraries for information on how to define and manage the shared libraries.

If the default permissions for a shared library (union of the permissions defined in the java.policy file, the app.policy file and the library.policy file) are enough, no action is required. The default library policy is picked up automatically. If a specific change is required to share a library in the cell, update the library.policy file.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

Important: Do not place the codebase keyword or any other keyword after the grant keyword. The Signed By keyword and the Java Authentication and Authorization Service (JAAS) Principal keyword are not supported in the library.policy file. The Signed By keyword is supported in the following policy files: java.policy, server.policy, and client.policy. The JAAS Principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=*URL* where *URL* is the location of the authorization policy.

1. Extract the policy file.
   a. From the command prompt, enter wsadmin wsadmin> set obj [$AdminConfig extract cells/*cell_name*/nodes/*node_name*/library.policy c:/temp/test/library.policy]
2. Edit the extracted library.policy file with the Policy Tool. For more information, see "Using PolicyTool to edit policy files" on page 1216.
3. Check in the policy file.
   a. Enter the following from a command prompt wsadmin> $AdminConfig checkin cells/*cell_name*/nodes/*node_name*/library.policy c:/temp/test/library.policy $obj.

An updated library.policy is applied to shared libraries after the servers restart.

The union of the permission contained in the java.policy file, the app.policy file, and the library.policy file are applied to the shared libraries. The library.policy file is managed by configuration and file replication services. Changes made in the file are replicated to other nodes in the Network Deployment cell.

The library.policy file supplied by WebSphere Application Server resides at: *install_root*/config/cells/*cell_name*/nodes/*node_name*/library.policy, contains an empty permission entry as a default. For example,

```
grant {
 };
```

If the shared library in a cell requires permissions that are not defined as defaults in the java.policy file, app.policy file and the library.policy file, update the library.policy file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
 java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the library.policy file, for example: `grant codeBase "file:<user client installed location>" { permission java.io.FilePermission "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };`

to decide whether to add a permission, refer to “Access control exception” on page 1213.

Restart the related Java processes for the changes in the library.policy file to become effective.

Adding the was.policy file to applications:

When Java 2 security is enabled for a WebSphere Application Server, all the applications that run on that WebSphere Application Server undergo a security check before accessing system resources. An application might need a was.policy file if it accesses resources that require more permissions than those granted in the default app.policy file. By default, the product security reads an app.policy file that is located in each node and grants the permissions in the app.policy file to all the applications. Include any additional required permissions in the was.policy file. The was.policy file is only required if an application requires additional permissions.

The default policy file for all applications is specified in the app.policy file. This file is provided by the product security, is common to all applications, and should not be changed. Add any new permissions required for an application in the was.policy file.

The app.policy file is located in the *install_root*/config/cells/*cell_name*/nodes/*node_name* directory. The contents of the app.policy file follow:

Attention: In the following code sample, the two permissions that are required by JavaMail were split into two lines each due to the width of the printed page.

```
// The following permissions apply to all the components under the application.
grant codeBase "file:${application}" {
   // The following are required by JavaMail
   permission java.io.FilePermission "
         ${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
   permission java.io.FilePermission "
```

```
        ${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
};

  // The following permissions apply to all utility .jar files  (other
  //  than enterprise beans JAR files) in the application.
grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to connector resources within the application
grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to all the Web modules (.war files)
// within the application.
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
        //  where "was.module.path" is the path where the Web module is
        //  installed. Refer to Dynamic policy concepts for other symbols.
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};


// The following permissions apply to all the EJB modules within the application.
grant codeBase "file:${ejbComponent}" {
 permission java.lang.RuntimePermission "queuePrintJob";
 permission java.net.SocketPermission "*", "connect";
 permission java.util.PropertyPermission "*", "read";
};
```

If additional permissions are required for an application or for one or more modules of an application, use the `was.policy` file for that application. For example, use codeBase of ${application} and add required permissions to grant additional permissions to the entire application. Similarly, use codeBase of ${webComponent} and ${ejbComponent} to grant additional permissions to all the Web modules and all the enterprise bean (EJB) modules in the application. You can assign additional permissions to each module (`.war` file or `.jar` file) as shown in the following example.

An example of adding extra permissions for an application in the `was.policy` file:

Attention: In the following code sample, the permission for the EJB module was split into two lines due to the width of the printed page.

```
// grant additional permissions to a Web module
grant codeBase " file:aWebModule.war" {
 permission java.security.SecurityPermission "printIdentity";
};

// grant additional permission to an EJB module
grant codeBase "file:aEJBModule.jar"  {
     permission java.io.FilePermission "
```

```
        ${user.install.root}${/}bin${/}DefaultDB${/}-" ."read.write,delete";
    // where, ${user.install.root} is the system property whose value is
    // located in the <install_root> directory.
};
```

1. Create a `was.policy` file using the policy tool. For more information on using the policy tool, see "Using PolicyTool to edit policy files" on page 1216
2. Add the required permissions in the `was.policy` file using the policy tool.
3. Place the `was.policy` file in the application enterprise archive (EAR) file under the `META-INF` directory. Update the application EAR file with the newly created `was.policy` file by using the **jar** command.
4. Verify that the `was.policy` file is inserted, and start an assembly tool. For more information, see Starting an assembly tool
   a. Verify that the `was.policy` file in the application is syntactically correct. In an assembly tool, right-click the enterprise application module and click **Run Validation**.

An application EAR file is now ready to run when Java 2 security is enabled.

This step is required for applications to run properly when Java 2 security is enabled. If the `was.policy` file is not created and it does not contain required permissions, the application might not access system resources.

The symptom of the missing permissions is the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When an application program receives this exception and adding this permission is justified, include the permission in the `was.policy` file, for example,

```
grant codeBase "file:${application}" { permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

The previous two lines are one continuous line.

Install the application.

Configuring static policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See the "Java 2 security policy files" on page 1217 article for the list of available policy files supported by WebSphere Application Server Version 6.

There are two types of policy files supported by WebSphere Application Server Version 6, dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application's permissions.

Policy file name	Description
`java.policy`	Contains default permissions for all of the Java programs on the node. This file seldom changes.
`server.policy`	Contains default permissions for all of the WebSphere Application Server programs on the node. This files is rarely updated.

Policy file name	Description
client.policy	Contains default permissions for all of the applets and client containers on the node.

The static 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 the other machine.

1. Identify the policy file to update.
   - If the permission is required only by an application, update the dynamic policy file. Refer to "Configuring Java 2 security policy files" on page 1222.
   - If the permission is required only by applets and client containers, update the client.policy file. Refer to "Configuring client.policy files" on page 1239.
   - If the permission is required only by WebSphere Application Server (servers, agents, managers and application servers), update the server.policy file. Refer to "Configuring server.policy files" on page 1238.
   - If the permission is required by all of the Java programs running on the Java virtual machine (JVM), update the java.policy file. Refer to "Configuring java.policy files."
2. Stop and restart the WebSphere Application Server.

The required permission is granted for all of the Java programs running with the restarted JVM.

If Java programs on a node require permissions, the policy file needs updating. If the Java program that required the permission is not part of an enterprise application, update the static policy file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate policy file, for example:

```
grant codeBase "file:<user client installed location>" {
  permission java.io.FilePermission
  "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar",
  "read";
};
```

To decide whether to add a permission, refer to "Access control exception" on page 1213.

Configuring java.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See "Java 2 security policy files" on page 1217 for the list of available policy files supported by WebSphere Application Server Version 6.0.x. The java.policy file is a global default policy file shared by all of the Java programs running in the Java Virtual Machine (JVM) on the node. Modifying this file is not recommended.

If a specific change is required to some of the Java programs on a node and the java.policy file requires updating, modify the java.policy file with policy tool. For more information, see "Using PolicyTool to edit policy files" on page 1216. A change to the java.policy file is local for the node. The default Java policy is picked up automatically. Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

An updated java.policy file is applied to all the Java programs running in all the JVMs on the local node. Restart the programs for the updates to take effect

The java.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 the other machine. The java.policy file supplied by WebSphere Application Server is located at *install_root*/java/jre/lib/security/java.policy. It contains these default permissions.

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/*" {
        permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
        // Allows any thread to stop itself using the java.lang.Thread.stop()
        // method that takes no argument.
        // Note that this permission is granted by default only to remain
        // backwards compatible.
        // It is strongly recommended that you either remove this permission
        // from this policy file or further restrict it to code sources
        // that you specify, because Thread.stop() is potentially unsafe.
        // See "http://java.sun.com/notes" for more information.
        // permission java.lang.RuntimePermission "stopThread";

        // allows anyone to listen on un-privileged ports
        permission java.net.SocketPermission "localhost:1024-", "listen";

        // "standard" properties that can be read by anyone

        permission java.util.PropertyPermission "java.version", "read";
        permission java.util.PropertyPermission "java.vendor", "read";
        permission java.util.PropertyPermission "java.vendor.url", "read";
        permission java.util.PropertyPermission "java.class.version", "read";
        permission java.util.PropertyPermission "os.name", "read";
        permission java.util.PropertyPermission "os.version", "read";
        permission java.util.PropertyPermission "os.arch", "read";
        permission java.util.PropertyPermission "file.separator", "read";
        permission java.util.PropertyPermission "path.separator", "read";
        permission java.util.PropertyPermission "line.separator", "read";

        permission java.util.PropertyPermission "java.specification.version", "read";
        permission java.util.PropertyPermission "java.specification.vendor", "read";
        permission java.util.PropertyPermission "java.specification.name", "read";

        permission java.util.PropertyPermission "java.vm.specification.version","read";
        permission java.util.PropertyPermission "java.vm.specification.vendor","read";
        permission java.util.PropertyPermission "java.vm.specification.name", "read";
        permission java.util.PropertyPermission "java.vm.version", "read";
        permission java.util.PropertyPermission "java.vm.vendor", "read";
        permission java.util.PropertyPermission "java.vm.name", "read";
       };
```

If some Java programs on a node require permissions that are not defined as defaults in the java.policy file, then consider updating the java.policy file. Most of the time, other policy files are updated instead of the java.policy file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `java.policy`file, for example:

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to “Access control exception” on page 1213.

Restart all of the Java processes for the updated `java.policy` file to take effect.

Configuring server.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 1217 for the list of available policy files supported by WebSphere Application Server Version 6.0.x. The `server.policy` file is a default policy file shared by all of the WebSphere servers on a node. 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 replicate to the other machine.

If the default permissions for a server (the union of the permissions defined in the `java.policy` file and the `server.policy` file) are enough, no action is required. The default server policy is picked up automatically. If a specific change is required to some of the server programs on a node, update the `server.policy` file with the Policy Tool. Refer to the “Using PolicyTool to edit policy files” on page 1216 article to edit policy files. Changes to the `server.policy` file are local for the node. Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

An updated `server.policy` file is applied to all the server programs on the local node. Restart the servers for the updates to take effect.

If you want to add permissions to an application, use the `app.policy` file and the `was.policy` file.

When you do need to modify the `server.policy` file, locate this file at:
install_root`/properties/server.policy`. This file contains these default permissions:

```
// Allow to use sun tools
grant codeBase "file:${java.home}/../lib/tools.jar" {
  permission java.security.AllPermission;
};

// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};

// Allow the WebSphere deploy tool all permissions
```

```
grant codeBase "file:${was.install.root}/deploytool/-" {
  permission java.security.AllPermission;
};
```

If some server programs on a node require permissions that are not defined as defaults in the `server.policy` file and the `server.policy` file, update the `server.policy` file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `server.policy` file, for example:

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to “Access control exception” on page 1213.

Restart all of the Java processes for the updated `server.policy` file to take effect.

Configuring client.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 1217 for the list of available policy files supported by WebSphere Application Server The `client.policy` file is a default policy file shared by all of the WebSphere Application Server client containers and applets on a node. The union of the permissions contained in the `java.policy` file and the `client.policy` file are given to all of the WebSphere client containers and applets running on the node. The `client.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 replicate to the other machine. The `client.policy` file supplied by WebSphere Application Server is located at *install_root*`/profiles/`*profile_name*`/properties/client.policy`. It contains these default permissions:

```
grant codeBase "file:${java.home}/lib/ext/*" {
  permission java.security.AllPermission;
};
// IBM Developer Kit, Java Technology Edition classes
grant codeBase "file:${java.home}/lib/ext/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${java.home}/../lib/tools.jar" {
  permission java.security.AllPermission;
};
// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/installedConnectors/-" {
  permission java.security.AllPermission;
```

```
};
// J2EE 1.3 permissions for client container WAS applications
// in $WAS_HOME/installedApps
grant codeBase "file:${was.install.root}/installedApps/-" {
  //Application client permissions
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";
};
// J2EE 1.3 permissions for client container - expanded ear file code base
grant codeBase "file:${com.ibm.websphere.client.applicationclient.archivedir}/-"
 {
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";
};
// For MQ Series
grant codeBase "file:${mq.install.root}/java/*" {
  permission java.security.AllPermission;
};
```

1. If the default permissions for a client (union of the permissions defined in the `java.policy` file and the `client.policy` file) are enough, no action is required. The default client policy is picked up automatically.
2. If a specific change is required to some of the client containers and applets on a node, modify the `client.policy` file with the policy tool. Refer to “Using PolicyTool to edit policy files” on page 1216, to edit policy files. Changes to the `client.policy` file are local for the node.

All of the client containers and applets on the local node are granted the updated permissions at the time of execution.

If some client containers or applets on a node require permissions that are not defined as defaults in the `java.policy` file and the default `client.policy` file, update the `client.policy` file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines of sample code are one continuous line, but extended beyond the width of the page.

When a client program receives this exception and adding this permission is justified, add a permission to the `client.policy` file, for example, grant codebase ″file:*user_client_installed_location*″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.ja″, ″read″; };.

To decide whether to add a permission, refer to “Access control exception” on page 1213.

Close and restart the browser. You also must restart the client application if you have one.

Migrating Java 2 security policy:

Previous WebSphere Application Server releases

WebSphere Application Server uses the Java 2 security manager in the server run time 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 an undesirable operation for an application server.

To support Java 2 security properly, all the server run time 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 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 required by the server run time. This is due to the design and algorithm used by Java 2 security to enforce permission checks. Please refer to the Java 2 security check permission algorithm.

The following two permissions are enforced by the WebSphere Java 2 security manager (hard coded):
- `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 run time 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 run time 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 Version 6.0.x, which means all permissions are enforced. The default Java 2 security policy for enterprise application is the recommended permission set defined by the Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 specification. Refer to the *install_root*`/profiles/`*profile_name*`/config/cells/`*cell_name*`/nodes/`*node_name*`/app.policy` file for the default Java 2 security policy granted to enterprise applications. This is a much more stringent policy compared to previous releases.

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

Note: The default Java 2 security policy for enterprise applications is much more stringent and all permissions are enforced in WebSphere Application Server Version 6.0.x. It might fail because the application code does not have the necessary permissions granted where system resources (such as file I/O for example) 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 **Enforce Java 2 Security** option on the Global security settings 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). It contains both system permissions (permissions granted to the Java Virtual Machine (JVM) and the product server run time) 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 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 the WebSphere Application Server (action might be required). This is deprecated; superseded by `${user.install.root}` and `${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

There is no easy way of migrating the Java policy file to WebSphere Application Server Version 6.0.x automatically because there is 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 codebase is used instead of absolute codebase. There are many advantages to this process. The permissions defined in the `was.policy` file should only be granted to the specific enterprise application, while permissions in the `app.policy` file apply to all the enterprise applications running on the node where the `app.policy` file belongs. Refer to the “Java 2 security policy files” on page 1217 article 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 (the default is *install_root*`profiles` *profile_name*`/properties/java.policy`) for the `app1.ear` enterprise application and the system permissions (permissions granted to the JVM and product server run time). Default permissions are omitted for clarity:

```
// For product Samples
   grant codeBase "file:${install_root}/installedApps/app1.ear/-" {
     permission java.security.SecurityPermission "printIdentity";
     permission java.io.FilePermission "${install_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 one is not present) or update the `was.policy` for migrated applications in the configuration repository in (`profiles/`*`profile_name`*`config/cells/`*`cell_name`*`/applications/app.ear/deployments/app/META-INF/was.policy`) 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 one continuous line, but extended beyond the width of the page.
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 "Configuring the was.policy file" on page 1228.
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 `${was.install.root}profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/nodes/`*`node_name`*`/app.policy` file. This requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a pre-production 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 pre-production testing of the migrated enterprise application with Java 2 security enabled. **Hint**: Enable trace for the WebSphere Application Server Java 2 security manager in the pre-production testing environment (with trace string: `com.ibm.ws.security.core.SecurityManager=all=enabled`). This can be helpful in debugging the *AccessControlException* exception thrown 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 granted to the classes on the call stack when the exception is thrown. For more information, see "Access control exception" on page 1213.

   Note: Because the Java 2 security policy is much more stringent compared with previous releases, it is strongly advised that the administrator or deployer 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.

Configuring security with scripting

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 Simple Object Access Protocol (SOAP) connector:
  - If you use a 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
    ```

    You can find the default value for this property in the `soap.client.props` file. If you accept the default value and do not provide loginUserid and loginPassword values, 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.sh -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 global security using scripting

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 global security based on LocalOS registry.

- 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 global 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 global security based on the LocalOS registry, use the following procedure call:
  - Using Jacl:

    ```
    securityoff
    ```

  - Using Jython:

    ```
    securityoff()
    ```

Enabling and disabling Java 2 security using 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 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.

Deploying secured applications

Before you perform this task, verify that you have already designed, developed and assembled an application with all the relevant security configurations. For more information on these tasks refer to Developing secured applications and Assembling secured applications. In this context, deploying and installing an application are considered the same task.

Deploying applications that have security constraints (secured applications) is not much different than deploying applications that don't contain any security constraints. The only difference is that you might need to assign users and groups to roles for a secured application, which requires that you have the correct active registry. To deploy a newly secured application click **Applications > Install New Application** in the navigation panel on the left and follow the prompts. If you are installing a secured application, roles would have been defined in the application. If delegation was required in the application, RunAs roles also are defined.

One of the steps required to deploy secured applications is to assign users and groups to roles defined in the application. This task is completed as part of the step titled *Map security roles to users and groups*. This assignment might have already been done through an assembly tool. In that case you can confirm the mapping by going through this step. You can add new users and groups and modify existing information during this step.

If the applications support delegation, then a RunAs role is already defined in the application. If the delegation policy is set to **Specified Identity** (during assembly) the intermediary invokes a method using an identity setup during deployment. Use the RunAs role to specify the identity under which the downstream invocations are made. For example, if the RunAs role is assigned user ″bob″ and the client ″alice″ is invoking a servlet, with delegation set, which in turn calls the enterprise beans, then the method on the enterprise beans is invoked with ″bob″ as the identity. As part of the deployment process one of the steps is to assign or modify users to the RunAs roles. This step is titled ″Map RunAs roles to users″. Use this step to assign new users or modify existing users to RunAs roles when the delegation policy is set to Specified Identity.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the ″Map security roles to users and groups″ link during application installation and also during managing applications, as a link in the Additional properties section.

1. Click **Applications > Install New Application**. Complete the steps (non-security related) that are required prior to the step entitled **Map security roles to users and groups**.

   Note: Depending upon the configuration, System Authorization Facility (SAF) configuration will take precedence.
2. Map users to RunAs roles if RunAs roles exist in the application. For more information, see "Assigning users to RunAs roles" on page 1251.
3. Click **Correct use of System Identity** to specify RunAs roles if needed. Complete this action if the application has delegation set to use System Identity (applicable to enterprise beans only). System Identity uses the WebSphere Application Server security server ID to invoke downstream methods and should be used with caution as this ID has more privileges than other identities in terms of accessing WebSphere Application Server internal methods. This task is provided to make sure that the deployer is aware that the methods listed in the panel have System Identity set up for delegation and to correct them if necessary. If no changes are necessary, skip this task.
4. Complete the remaining (non-security related) steps to finish installing and deploying the application.

Once a secured application is deployed, verify that you can access the resources in the application with the correct credentials. For example, if your application has a protected Web module, make sure only the users that you assigned to the roles are able to use the application.

Assigning users and groups to roles

This topic describes how to assign users and groups to roles if you are using WebSphere Application Server authorization for Java 2 Platform, Enterprise Edition (J2EE) roles. If you are using System Authorization Facility (SAF) authorization for J2EE roles, this is done independently of the application deployment process. For more information, refer to “System Authorization Facility for role-based authorization” on page 855.

Before you perform this task:
- Secure the Web applications and EJB applications where new roles were created and assigned to Web and Enterprise JavaBeans (EJB) resources.
- Create all the roles in your application.
- Verify that you have properly configured the user registry that contains the users that you want to assign. It is preferable to have security turned on with the user registry of your choice before beginning this process.
- Make sure that if you change anything in the security configuration (for example, enable security or change the user registry) you save the configuration and restart the server before the changes become effective.

Because the default active user registry is Local OS, it is not necessary, although it is recommended, that you enable security if you want to use the Local OS user registry to assign users and groups to roles. You can enable security once the users and groups are assigned in this case. The advantage of enabling security with the appropriate registry before proceeding with this task is that you can validate the security setup (which includes checking the user registry configuration) and avoid any problems using the registry.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the Map security roles to users/groups link during application installation and also during application management, as a link in the Additional properties section.

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Click **Applications > Enterprise applications >** *application_name*.
3. Under Additional properties, click **Map security roles to users/groups**. A list of all the roles that belong to this application displays. If the roles already had users or special subjects (All Authenticated, Everyone) assigned, they display here.
4. To assign the special subjects, select either the **Everyone** or the **All Authenticated** option for the appropriate roles.
5. Click **Apply** to save any changes and then continue working with user or group roles.
6. To assign users or groups, select the role. You can select multiple roles at the same time, if the same users or groups are assigned to all the roles.
7. Click **Look up users** or **Look up groups**.
8. Get the appropriate users and groups from the registry by completing the **limit** (number of items) and the **Search String** fields and clicking **Search**. The **limit** field limits the number of users that are obtained and displayed from the registry. The pattern is a searchable pattern matching one or more users and groups. For example, `user*` lists users like user1, user2. A pattern of asterisk (*) indicates all users or groups.

   Use the limit and the search strings cautiously so as not to overwhelm the registry. When using large registries (like Lightweight Directory Access Protocol (LDAP)) where information on thousands of users and groups resides, a search for a large number of users or groups can make the system very slow and can make it fail. When there are more entries than requests for entries, a message displays on top of the panel. You can refine your search until you have the required list.
9. Select the users and groups to include as members of these roles from the **Available** field and click **>>** to add them to the roles.

10. To remove existing users and groups, select them from the **Selected** field and click **<<**. When removing existing users and groups from roles use caution if those same roles are used as RunAs roles.

    For example, if user1 is assigned to RunAs role, role1, and you try to remove user1 from role1, the administrative console validation does not delete the user since a user can only be a part of a RunAs role if the user is already in a role (User1 should be in role1 in this case) either directly or indirectly through a group. For more information on the validation checks that are performed between RunAs role mapping and user and group mapping to roles, see the “Assigning users to RunAs roles” on page 1251 section.
11. Click **OK**. If there are any validation problems between the role assignments and the RunAs role assignments the changes are not committed and an error message indicating the problem displays at the top of the panel. If there is a problem, make sure that the user in the RunAs role is also a member of the regular role. If the regular role contains a group which contains the user in the RunAs role, make sure that the group is assigned to the role using the administrative console. Follow steps 4 and 5. Avoid using the Application Server Toolkit or any other manual process where the complete name of the group, host name, group name, or distinguished name (DN) is not used.

The user and group information is added to the binding file in the application. This information is used later for authorization purposes.

This task is required to assign users and groups to roles, which enables the correct users and groups to access a secured application. If you are installing an application, complete your installation. Once the application is installed and running you can access your resources according to the user and group mapping you did in this task. If you are managing applications and have modified the users and groups to role mapping, make sure you save, stop and restart the application so that the changes become effective. Try accessing the J2EE resources in the application to verify that the changes are effective.

Security role to user and group selections:

Use this page to select users and groups for security roles.

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

While using the Install New Application Wizard, prompts appear to help you map security roles to users. You also can configure security roles to user mappings of deployed applications. Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups and roles are defined when an application is installed or configured.

You also can select role to user and group mappings while you are deploying applications. After deployment in **Additional Properties**, click **Map Security roles to users** to change user and group mappings to a role.

Look up users:

Specifies whether the server looks up selected users.

Choose the role by selecting the check box beside the role and clicking **Lookup users**. Complete the **Limit** and the **Pattern** fields. The **Limit** field contains the number of entries that the search function returns. The **Pattern** field contains the search pattern used for searching entries. For example, `bob*` searches all users or groups starting with `bob`. A limit of zero returns all the entries that match the pattern. Use this value only when a small number of users or groups match this pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

Look up groups:

Specifies whether the server looks up selected groups.

Choose the role by selecting the check box beside the role and clicking **Lookup groups**. Complete the **Limit** and the **Pattern** fields. The **Limit** field contains the number of entries that the search function returns. The **Pattern** field contains the search pattern used for searching entries. For example, `bob*` searches all users or groups starting with `bob`. A limit of zero returns all the entries that match the pattern. Use this value only when a small number of users or groups match this pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

Role:

Specifies user roles.

A number of administrative roles are defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

Monitor
This role is the least privileged. A user can view the server configuration and its current state.

Configurator
This role has the monitor privilege plus the ability to change the server configuration.

Operator
This role has the monitor privilege plus the ability to change the run-time state, such as starting or stopping services

Administrator
This role has the operator privileges plus the configurator privileges.

Range Monitor, Configurator, Operator, Administrator

Everyone:

Specifies to authenticate everyone.

Range Monitor, Configurator, Operator, Administrator

All authenticated:

Range Monitor, Configurator, Operator, Administrator

Mapped users:

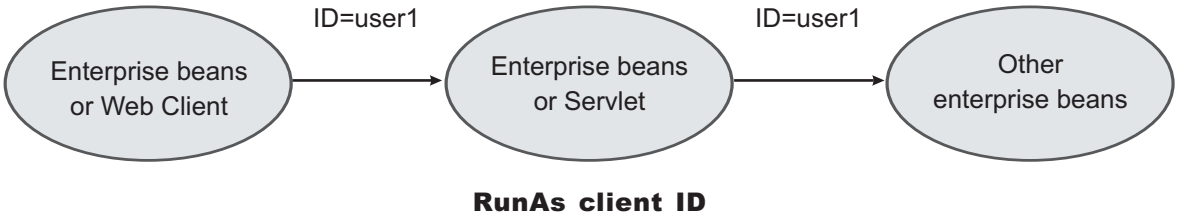
Mapped groups:

Delegations

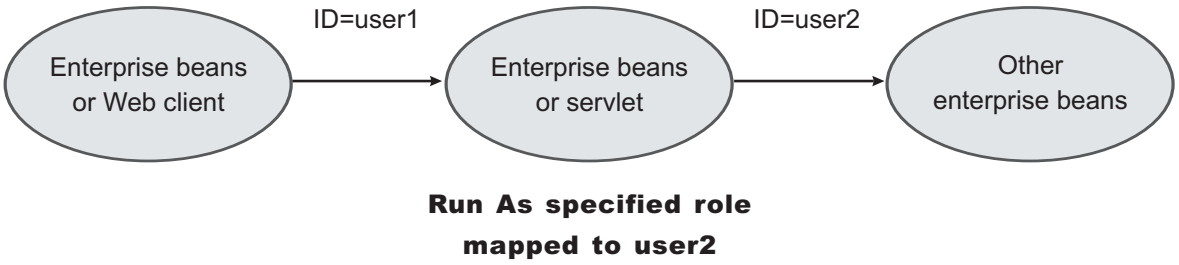
Delegation is a process security identity propagation from a caller to a called object. As per the J2EE specification, a servlet and enterprise beans can propagate either the client (remote user) identity when invoking enterprise beans or they can use another specified identity as indicated in the corresponding deployment descriptor.

The IBM extension supports Enterprise JavaBeans (EJB) to propagate to the server ID when invoking other entity beans. There are three types of delegations:
- Delegate (RunAs) Client Identity
- Delegate (RunAs) Specified Identity
- Delegate (RunAs) System Identity

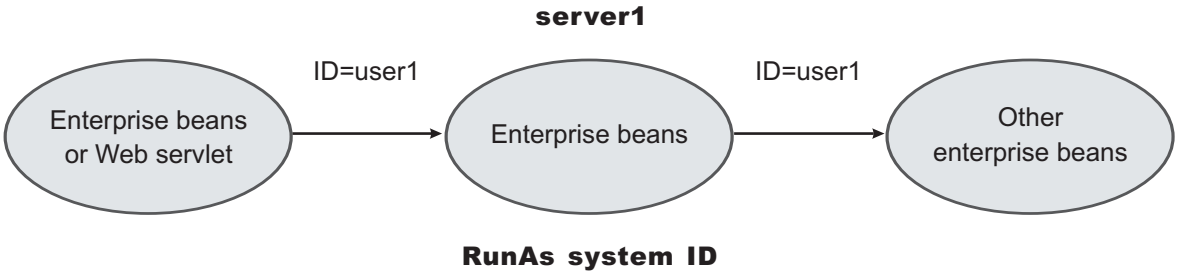
Delegate (RunAs) Client Identity



Delegate (RunAs) Specified Identity



Delegate (RunAs) System Identity

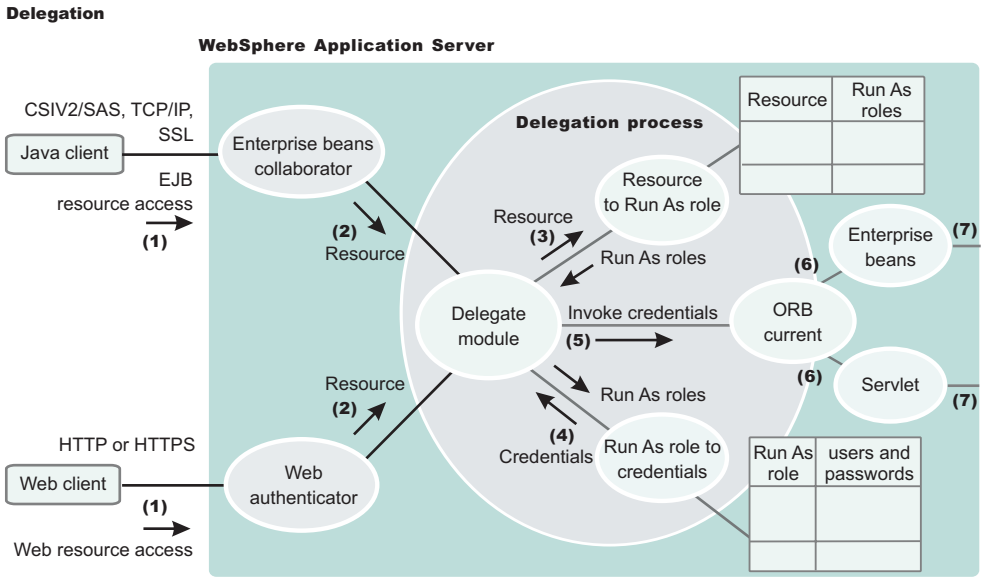


The EJB specification only supports delegation (RunAs) at the EJB level. But an IBM extension allows EJB method level RunAs specification. Method EJB method level runAs specification allows one to specify a different RunAs role for different methods within the same enterprise beans.

The RunAs specification is detailed in the deployment descriptor (the `ejb-jar.xml` file in the EJB module and the `web.xml` file in the Web module). The IBM extension to the RunAs specification is included in the `ibm-ejb-jar-ext.xmi` file.

There is also an IBM specific binding file 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 run time during application startup. The following figure illustrates the delegation mechanism as implemented in the WebSphere Application Server security model.



Delegation Process

There are two tables that 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 figure.

Assigning users to RunAs roles

Before you perform this task, complete the following tasks:
- Secure the Web applications and EJB applications where new RunAs roles were created and assigned to Web and EJB resources.
- Create all the RunAs roles in your application. The user in the RunAs role can only be entered if that user or a group to which that user belongs is already part of the regular role.

- Assign users and groups to security roles. Refer to “Assigning users and groups to roles” on page 1247 for more information.
- Verify that the user registry requirements are met. These requirements are the same as those discussed in the same as in the case of “Assigning users and groups to roles” on page 1247 task. For example, if role1 is a role that is also used as a RunAs role, then the user, user1, can be added to the RunAs role. The administrative console checks this logic when **Apply** or **OK** is clicked. If the check fails, the change is not made and an error message displays at the top of the panel.

If the special subjects ″Everyone″ or ″All Authenticated″ are assigned to a role, then no check takes place for that role.

The checking is done every time **Apply** in this panel is clicked or when **OK** is clicked in the **Map security roles to users/groups** panel. The check verifies that all the users in all the RunAs roles do exist directly or indirectly (through a group) in those roles in the **Map security roles to users/groups** panel. If a role is assigned both a user and a group to which that user belongs, then either the user or the group (not both) can be deleted from **Map security roles to users/groups** panel.

If the RunAs role user belongs to a group and if that group is assigned to that role, make sure that the assignment of this group to the role is done through administrative console and not through an assembly tool or any other method. When using the administrative console, the full name of the group is used (for example, `hostname\groupName` in windows systems, and distinguished names (DN) in Lightweight Directory Access Protocol (LDAP)). During the check, all the groups to which the RunAs role user belongs are obtained from the registry. Since the list of groups obtained from the registry are the full names of the groups, the check works correctly. If the short name of a group is entered using an assembly tool (for example, `group1` instead of `CN=group1, o=myCompany.com`) then this check fails.

These steps are common to both installing an application and modifying an existing application. If the application contains RunAs roles, you see the **Map RunAs roles to users** link during application installation and also during managing applications as a link in the **Additional properties** section at the bottom.

1. Click **Applications > Enterprise Applications >** *application_name*.
2. Under Additional properties, click **Map RunAs roles to users**. A list of all the RunAs roles that belong to this application displays. If the roles already had users assigned, they display here.
3. To assign a user, select the role. You can select multiple roles at the same time if the same user is assigned to all the roles.
4. Enter the user’s name and password in the designated fields. The user name entered can be either the short name (preferred) or the full name (as seen when getting users and groups from the registry).
5. Click **Apply**. The user is authenticated using the active user registry. If authentication is successful, a check is made to verify that this user or group is mapped to the role in the **Map security roles to users and groups** panel. If authentication fails, verify that the user and password are correct and that the active registry configuration is correct.
6. To remove a user from a RunAs role, select the roles and click **Remove**.

The RunAs role user is added to the binding file in the application. This file is used for delegation purposes when accessing J2EE resources. This step is required to assign users to RunAs roles so that during delegation the appropriate user is used to invoke the EJB methods.

If you are installing the application, complete installation. Once the application is installed and running you can access your resources according to the RunAS role mapping. Save the configuration.

If you are managing applications and have modified the RunAs roles to users mapping, make sure you save, stop and restart the application so that the changes become effective. Try accessing your J2EE resources to verify that the new changes are in effect.

EJB 2.1 method protection level settings:

Use this page to verify that all unprotected EJB 2.1 methods have the correct level of protection before you map users to roles.

To view this administrative console page, click **Applications > Install New Application**. While running the Install New Application Wizard, prompts appear to help you determine that all unprotected EJB 2.1 methods have the correct level of protection.

EJB Module:

Specifies the enterprise bean module name.

Data Type: String
Units: EJB module name

Module URI:

Specifies the Java archive (JAR) file name.

Data Type: String
Units: JAR file name

Method protection:

Specifies the level of protection assigned to the EJB module.

A selected box means to *Deny All* and that the method is completely protected.

Data Type: Check box
Default: Cleared
Range: Yes or No

RunAs roles to users mapping:

Use this page to map RunAs roles to users. You can change the RunAs settings after an application deploys.

To view this administrative console page, click **Applications > Install New Application**. While running the application installation wizard, prompts appear to help you map RunAs roles to users. You can change the RunAs roles to users mappings for deployed applications by completing the following steps:

1. Click **Applications > Enterprise Applications >** *application_name*.
2. Under Additional properties, click **Map RunAs roles to users**.

The enterprise beans you are installing contain predefined RunAs roles. RunAs roles are used by enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean.

User name:

Specifies a user name for the RunAs role user.

This user already maps to the role specified in the Mapping users and groups to roles panel. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role.

Data type: String

Password:

Specifies the password for the RunAs user.

Data type: String

Confirm password:

Specifies the confirmed password of the administrative user.

Data type String

Role:

Specifies administrative user roles.

A number of administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the web based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

Monitor
This role is the least privileged. A user can view the server configuration and its current state.

Configurator
This role has the monitor privilege plus the ability to change the server configuration.

Operator
This role has the monitor privilege plus the ability to change the run-time state, such as starting or stopping services

Administrator
This role has the operator privileges plus the configurator privileges.

Updating and redeploying secured applications

Before you perform this task, secure Web applications, secure EJB applications, and deploy them in WebSphere Application Server. This section addresses the way to update existing applications.

1. Use the administrative console to modify the existing users and groups mapping to roles. For information on the required steps, see “Assigning users and groups to roles” on page 1247.
2. Use the administrative console to modify the users for the RunAs roles. For information on the required steps, see “Assigning users to RunAs roles” on page 1251.
3. Complete the changes and save them.
4. Stop and restart the application for the changes to become effective.
5. Use the an assembly tool. For more information, see Assembling applications.
6. Use an assembly tool to modify roles, method permissions, auth-constraints, data-constraints and so on. For more information, see Assembling applications.
7. Save the Enterprise Archive (EAR) file, uninstall the old application, deploy the modified application and start the application to make the changes effective.

The applications are modified and redeployed. This step is required to modify existing secured applications.

If information about roles is modified make sure you update the user and group information using the administrative console. Once the secured applications are modified and either restarted or redeployed, make sure that the changes are effective by accessing the resources in the application.

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 application 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. 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 1260.
4. 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 1260. For more information on configured name bindings, refer to "Configured name bindings" on page 1262.
5. 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 JavaBeans (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.

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.

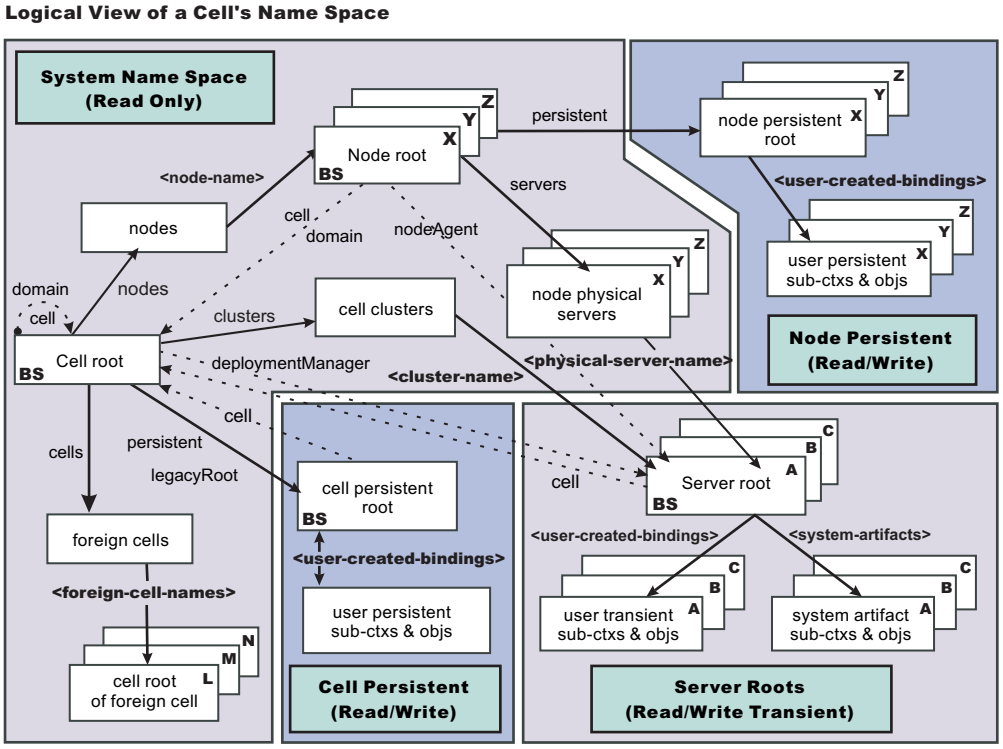


Figure 11. Name Space Logical View

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.

In WebSphere Application Server Network Deployment cells, the cell and node persistent areas can be read even if the deployment manager and respective node agent are not running. However, the deployment manager must be running to update the cell persistent segment, and a node agent must be running to update its respective node persistent segment.

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.

Distributing application objects among many server roots is a departure from previous WebSphere Application Server releases, where all system artifacts were bound under a single root. This change can affect the names that clients use to look up these objects.

Server-scoped 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.

The cell persistent area can be read even if the deployment manager is not running. However, the deployment manager must be running to update the cell persistent segment. Because every server contains its own copy of the cell persistent partition, any server can look up locally objects bound in the cell persistent partition.

An important role of the cell persistent root is as the initial context for clients running in previous WebSphere Application Server versions. If you want to access an enterprise bean by WebSphere Application Server v4.0.x and 3.5.x clients, you must ensure that a binding for it has been added to the cell persistent root. You can configure these additional bindings as cell-scoped bindings.

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.

The node persistent area for a node can be read from any server in the node even if the respective node agent is not running. However, the node agent must be running to update the node persistent area, or for any server outside the node to read from that node persistent partition. Because every server in a node contains its own copy of the node persistent partition for its node, any server in the node can look up locally objects bound in that node persistent partition.

Unlike the cell persistent root, the node persistent root plays no special role in interoperability with WebSphere Application Server clients of previous releases. Node-scoped 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 V6 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.
- **WebSphere Application Server JNDI interface implementation prior to V5**

  WebSphere Application Server clients running in releases prior to WebSphere Application Server V5 by default use WebSphere Application Server's v4.0 CosNaming JNDI plug-in implementation. The default initial context for clients of this type is the cell persistent root, also known as the *legacy root*.
- **Other JNDI implementation**

Some applications can perform name space lookups with a non-WebSphere Application Server CosNaming JNDI plug-in implementation. Assuming the key **NamingContext** 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 **NamingContext**. The initial context in this case is the cell root.

Lookup names support in deployment descriptors and thin clients

Server 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 is a departure from previous versions of WebSphere Application Server, where these objects were all bound under a single root context. This section 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 speaking, 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 Sun's 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 Sun's 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 interoperability with JNDI clients running on previous versions of WebSphere Application Server. Additionally, you can configure cell-scoped bindings to create a fixed qualified name for server objects.

Scope

You can configure a binding at one of the following three scopes: cell, node, or server. 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. If the target server of a server-scoped binding is a cluster, the binding is created under the server root context of each cluster member.

Note: The term *server* includes clusters and can be used interchangeably with the term *cluster* with respect to configured bindings. When applied to a cluster, a server-scoped binding is created in the server root for all member servers.

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 or server, or if you do not want the binding to be associated with any specific node 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 if you want to configure a binding with the same name on different servers which resolve to different objects, a server-scoped binding would be appropriate. Note that two servers 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.

This type of binding is of special significance because you can use it to provide interoperability with WebSphere Application Server v3.5.x and v4.0.x JNDI clients. The default initial context for these earlier clients is the cell persistent root, which is different from the initial context of the server root for WebSphere Application Server V5 JNDI clients. If you migrate an application to the current release, you can configure an EJB binding at the cell scope so that the lookup names for the enterprise bean do not change for clients still running in a earlier WebSphere Application Server version.

A cell-scoped EJB binding is also 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.

Note: WebSphere Application Server v3.5.x clients cannot access this type of binding .

String: String constant

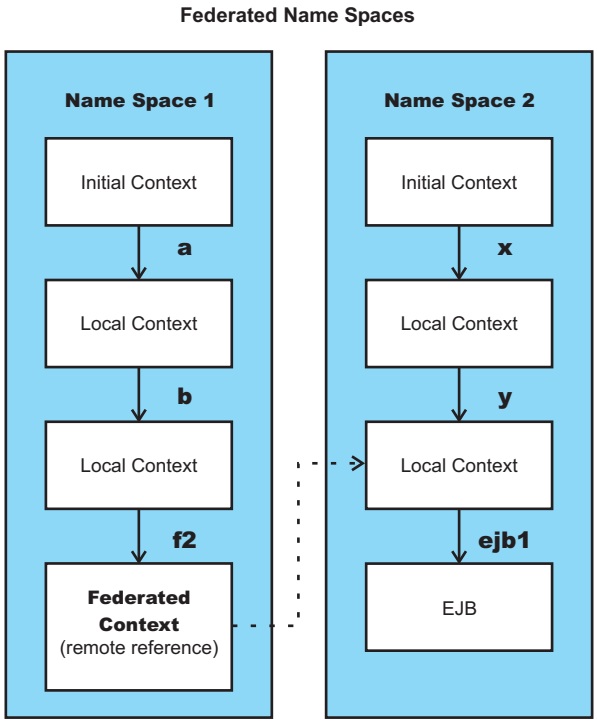
You can configure a binding of a string constant. The following data is required to configure a string constant binding:

- 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 JavaBeans (EJB) home bound under the name `ejb1`, the EJB home could 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 WebSphere Application Server's initial context factory to obtain the reference to the federated context. If you use some other initial context factory implementation, you either may not be able to create the binding, or the level of transparency may 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 may result. If you want to federate WebSphere Application Server name spaces, you should use servers running under the Network Deployment or Enterprise packages of WebSphere Application Server.

Name space bindings

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 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 to provide interoperability with JNDI clients running on previous versions of the WebSphere Application Server.

There are four different kinds of bindings that you can configure:
- Enterprise JavaBeans (EJB)
- CORBA object
- Indirect Lookup
- String

Naming and directories: Resources for learning

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

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

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.

   Note: All of these 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 and viewing name space bindings

To view or configure an EJB, CORBA, Indirect lookup or string name space binding, complete the following:

1. In the administrative console, click **Environment > Manage Name Space Bindings**.
2. Select the desired scope by entering in a node name for node-scoped bindings, or a node name and server name for server-scoped bindings, and click **Apply**.
3. To create a new binding, click **New** and follow the instructions. To edit a previously created binding, click the binding you want to edit and proceed to the next step.
4. Edit the **Binding identifier**, the **Name in name space**, and the **String value** fields as desired.

   Note: All of these fields are required.

5. Click **Finish** to register the changes.

String binding settings

Use this page to configure a new string binding or to view or edit an existing string binding.

To view this administrative console page, click **Environment** > **Naming** > **Name Space Bindings** > *string_namespace_binding*.

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. Possible choices are String, EJB, CORBA, and Indirect. This field 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.

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 administrative console page, click **Environment** > **Naming** > **Name Space Bindings** > *CORBA_namespace_binding*.

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. Possible choices are String, EJB, CORBA, and Indirect. This field 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 administrative console page, click **Environment** > **Naming** > **Name Space Bindings** > *indirect_lookup_namespace_binding*.

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. Possible choices are String, EJB, CORBA, and Indirect. This field 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.

EJB binding settings

Use this page to configure a new EJB binding, or to view or edit an existing EJB binding.

To view this administrative console page, click **Environment** > **Naming** > **Name Space Bindings** > *EJB_namespace_binding*.

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. Possible choices are String, EJB, CORBA, and Indirect. This field 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).

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 users in your collection. Use the buttons to control the selected users.

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.

Developing applications that use JNDI

References to EJB homes and other artifacts such as data sources are bound to the WebSphere 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 name space.

These 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 WebSphere's JNDI Context implementation is used.

WebSphere Application Server's 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's initial context factory. To set JNDI caching or name syntax properties which will be visible to WebSphere Application Server's initial context factory, follow the following steps.

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 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.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

This example below gets the default initial context. That is, no provider URL is passed to the javax.naming.InitialContext constructor. The following section explains the process of determining the address of the bootstrap server to use to obtain the initial context.

```
...
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. The initial context returned in the various environments are listed below:

- Thin client: The server root context of the server running on the local host at port 2809.
- Pure client:
  - The context specified by the java.naming.provider.url property passed to launchClient command with the -CCD command line parameter. The context usually will be 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 was specified, 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 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, which has changed from previous releases.

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.

- **Understanding the 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**

  Previous versions of WebSphere Application Server 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 previous releases of WebSphere Application Server, 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. Today, 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:
  1. The InitialContext constructor. This does not apply to the above example since the example uses the empty InitalContext constructor.
  2. 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.
  3. `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 may 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: 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.

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
Cell Persistent Root	NameServiceCellPersistentRoot

Root Context	CORBA Object URL Object Key
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 which 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. The examples below show lookups from each type of application. 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. An example of a lookup with a corbaname URL is also included in this section.

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.

```
// 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

The example below shows a lookup of a JavaMail resource. The actual lookup name is determined by the application's deployment descriptors.

```
// Get the initial context as shown above
...
Session session =
     (Session) initialContext.lookup("java:comp/env/mail/MailSession");
```

JNDI interoperability considerations

This section explains considerations to take into account when interoperating with WebSphere Application Server V4.0 and with non-WebSphere Application Server JNDI clients. Also, the way resources from MQSeries must be bound to the name space changed after V4.0 and is described below.

Interoperability with WebSphere Application Server V4.0

- **EJB clients running on WebSphere Application Server V4.0 accessing EJB applications running on WebSphere Application Server V5 or V6**

  Applications migrated from previous versions of WebSphere Application Server may still have clients still running in a previous release. The default initial JNDI context for EJB clients running on previous versions of WebSphere Application Server is the cell persistent root (legacy root). The home for an enterprise bean deployed in version 5 or 6 is bound to its server's server root context. In order for the EJB lookup name for down-level clients to remain unchanged, configure a binding for the EJB home under the cell persistent root.

- **EJB clients running on WebSphere Application Server V5 or V6 accessing EJB applications running on WebSphere Application Server V4.0 servers**

  The default initial context for a WebSphere Application Server V4.0 server is the correct initial context. Simply look up the JNDI name under which the EJB home is bound.

  Note: To enable WebSphere Application Server V5 or V6 clients to access version 4.x servers, the down-level installations must have e-fix PQ60074 installed.

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 will be 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, as shown below.

- **Single server**

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

  According to the URL above, the bootstrap host and port are myHost and 2809, and the enterprise bean is installed in a server **server1** in node **node1** and bound in that server under the name **myEJB**.

- **Server cluster**

  ```
  initialContext.lookup(
      "corbaname:iiop:myHost:2809#cell/clusters/myCluster/myEJB");
  ```

  According to the URL above, the bootstrap host and port are **myHost** and **2809**, and the enterprise bean is installed in a server cluster named **myCluster** and bound in that cluster under the name **myEJB**.

  The above lookup will work with any name server bootstrap host and port configured in the same cell.

  The above lookup will also work if the bootstrap host and port belongs to a member of the cluster itself. To avoid a single point of failure, the bootstrap server host and port for each cluster member could 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

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 which must interoperate with CORBA applications may need to use INS name syntax to represent names in string format. The name syntax property may 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 will 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");
...
```

Developing applications that use CosNaming (CORBA Naming interface)

CORBA clients can perform naming operations on WebSphere name servers through the CosNaming interface. The following examples show how to obtain an ORB instance and an initial context as well as how to look up an EJB home.

Note: To enable WebSphere Application Server Version 6 or 5.x clients to access Version 4.0.x servers, the earlier installations must have e-fix PQ60074 installed.

1. Get an initial context.
2. Perform desired CosNaming operations.

Example: Getting an initial context with CosNaming

In the WebSphere Application Server, an initial context is obtained from a bootstrap server. The address for the bootstrap server consists of a host and port. To get an initial context, you must know the host and port for the server that is used as the bootstrap server.

Obtaining an initial context consists of two basic steps:
1. Obtain an ORB reference
2. Invoke a method on the ORB to obtain the initial reference

These steps are now explained in more detail.

Obtaining an ORB reference: Pure CosNaming clients, that is clients that are not running in a server process, must create and initialize an ORB instance with which to obtain the initial context. CosNaming clients which run in server processes can obtain a reference to the server ORB with a JNDI lookup. The following examples illustrate how to create and initialize a client ORB and how to obtain a server ORB reference.

Creating a client ORB instance

To create an ORB instance, invoke the static method, org.omg.CORBA.ORB.init. The init method requires a property set to the name of the ORB class you want to instantiate. An ORB implementation with the class name com.ibm.CORBA.iiop.ORB is included with the WebSphere Application Server. The WebSphere Application Server ORB recognizes additional properties with which you can specify initial references.

The basic steps for creating an ORB are as follows:
1. Create a Properties object.
2. Set the ORB class property to WebSphere Application Server's ORB class.
3. If the bootstrap server is INS-compliant, set the initial reference properties. If the bootstrap server is not INS-compliant (meaning, WebSphere Application Server v4.0.x or earlier), set bootstrap host and port for bootstrap server.
4. Invoke ORB.init, passing in the Properties object.

Usage scenario

```
...
import java.util.Properties;
import org.omg.CORBA.ORB;
...
Properties props = new Properties();
props.put("org.omg.CORBA.ORBClass","com.ibm.ws390.orb.ORB");
props.put("com.ibm.CORBA.ORBInitRef.NameService",
      "corbaloc:iiop:myhost.mycompany.com:2809/NameService");
props.put("com.ibm.CORBA.ORBInitRef.NameServiceServerRoot",
      "corbaloc:iiop:myhost.mycompany.com:2809/NameServiceServerRoot");
ORB _orb = ORB.init((String[])null, props);
...
```

Notice the initial reference definitions for NameService and NameServiceServerRoot. The initial context returned for NameService depends on the type of bootstrap server. The key NameServiceServerRoot is a key introduced in WebSphere Application Server V5. For more information on initial contexts, see the section Initial Contexts.

Note: The properties com.ibm.CORBA.BootstrapHost and com.ibm.CORBA.BootstrapPort are deprecated. They are needed, however, to connect to WebSphere Application Servers of Version 4.0.x or earlier. The default bootstrap host is the local host and the default port is 2809.

Obtaining a reference to the server ORB

CosNaming clients which run in a server process can obtain a reference to the server ORB with a JNDI lookup on a java: name, shown as follows:

Usage scenario

```
...
import javax.naming.Context;
import javax.naming.InitialContext;
import org.omg.CORBA.ORB;
...
Context initialContext = new InitialContext();
ORB orb = (ORB) initialContext.lookup("java:comp/ORB");
...
```

Using an ORB reference to get an initial naming reference: There are two basic ways to get an initial CosNaming context. Both ways involve an ORB method invocation. The first way is to invoke the resolve_initial_references method on the ORB with an initial reference key. For this call to work, the ORB

must be initialized with an initial reference for that key. The other way is to invoke the string_to_object method on the ORB, passing in a CORBA object URL with the host and port of the bootstrap server. The following examples illustrate both approaches.

Invoking resolve_initial_references

Once an ORB reference is obtained, invoke the resolve_initial_references method on the ORB to obtain a reference to the initial context. The following code example invokes resolve_initial_reference on an ORB reference.

Usage scenario

```
...
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
...
// Obtain ORB reference as shown in examples earlier in this section
...
org.omg.CORBA.Object obj = _orb.resolve_initial_references("NameService");
NamingContextExt initCtx = NamingContextExtHelper.narrow(obj);
...
```

Note that the key NameService is passed to the resolve_initial_references method. Other initial context keys are registered in WebSphere Application Servers. For example, NameServiceServerRoot can be used to obtain a reference to the server root context in the bootstrap name server. For more information on the initial contexts registered in server ORBs, see the section Initial Contexts.

Invoking string_to_object with a CORBA object URL

You can use an INS-compliant ORB to obtain an initial context even if the ORB is not initialized with any initial references or bootstrap properties, or if those property settings are for a different server than the name server from which you want to obtain the initial context. To obtain an initial context by explicitly specifying the bootstrap name server, invoke the string_to_object method on the ORB, passing in a CORBA object URL which contains the bootstrap server host and port.

The code in the example below invokes the string_to_object method on an existing ORB reference, passing in a CORBA object URL which identifies the desired initial context.

Usage scenario

```
...
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
...
// Obtain ORB reference as shown in examples earlier in this section
...
org.omg.CORBA.Object obj =
 orb.string_to_object("corbaloc:iiop:myhost.mycompany.com:2809/NameService");
NamingContextExt initCtx = NamingContextExtHelper.narrow(obj);
...
```

Note that the key NameService is used in the corbaloc URL. Other initial context keys are registered in WebSphere Application Servers. For example, you can use NameServiceServerRoot to obtain a reference to the server root context in the bootstrap name server.

Using an existing ORB and invoking string_to_object with a CORBA object URL with multiple name server addresses to get an initial context: CORBA object URLs can contain more than one bootstrap server address. Use this feature when attempting to obtain an initial context from a server cluster. You can specify the 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. There is no guarantee of any particular order in which the address list will be processed. For example, the second bootstrap server address may be used to obtain the initial context even though the first bootstrap server in the list is available. An example of a corbaloc URL with multiple addresses follows.

```
...
import org.omg.CORBA.ORB;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
...
// Assume orb is an existing ORB instance
org.omg.CORBA.Object obj = orb.string_to_object(
   "corbaloc::myhost1:9810,:myhost1:9811,:myhost2:9810/NameService");
NamingContextExt initCtx = NamingContextExtHelper.narrow(obj);
...
```

Example: Looking up an EJB home with CosNaming

You can look up an EJB home or other CORBA object from a WebSphere Application Server name server through the CORBA CosNaming interface. You can invoke `resolve` or `resolve_str` on the initial context, or you can invoke `string_to_object` on the ORB. You can use a qualified name so that the name resolves regardless of which name server the lookup is executed on, or use an unqualified name that only resolves from the server root context on the name server that actually contains the object binding. (The qualified name traverses the federated system name space to the specified server root context.)

Qualified and unqualified names

Each application server contains a name server. System artifacts such as EJB homes are bound 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 use a qualified name. A qualified name can be a topology-based name, based on the name of the cluster or single server and node that contains the object. You can define fixed qualified names for objects. With qualified names, you can look up objects residing on different servers from the same initial context by traversing the system name space structure. Alternatively, you can use an unqualified name, but an unqualified name will only resolve using the name server associated with the object's application server.

CosNaming.resolve (and resolve_str) vs. ORB.string_to_object

If you have an initial context from any name server in a WebSphere Application Server cell, you can look up any CORBA object with a qualified name. You do not need additional host and port information for the target object's name server.

Alternatively, you can look up an object by invoking `string_to_object` on the ORB, passing in a corbaname URL. Typically, an IIOP type URL is specified, so the bootstrap address information required for an initial context must be contained in the URL. You can use a qualified or unqualified stringified name, but an unqualifed name resolves only if the initial context is from the name server in which the object is bound.

The following examples show CosNaming resolve operations using qualified topology-based lookup names and an unqualified lookup name.

CosNaming resolve operation using a qualified name: The topology-based qualified name for an object depends on whether the object is bound in a single server or a server cluster. Examples of each follow.

Single server

The following example shows the lookup of an EJB home that is running in a single server. The enterprise bean that is being looked up is running in the server, `MyServer`, on the node, `Node1`.

```
// Get the initial context as shown in the 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 name under which the EJB home is bound
org.omg.CORBA.Object ejbHome = initialContext.resolve_str(
  "cell/nodes/Node1/servers/MyServer/mycompany/accounting/AccountEJB");
accountHome =
  (AccountHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, AccountHome.class);
```

Server cluster

The following example shows a lookup of an EJB home that is running in a cluster. The enterprise bean being that is looked up is running in the cluster, Cluster1. The name can be resolved if any of the cluster members is running.

Usage scenario

```
// Get the initial context as shown in the 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 name under which the EJB home is bound
org.omg.CORBA.Object ejbHome = initialContext.resolve_str(
   "cell/clusters/Cluster1/mycompany/accounting/AccountEJB");
accountHome =
   (AccountHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, AccountHome.class);
```

ORB string_to_object operation using an unqualified stringified name: If the resolve operation is being performed on the name server that contains the object, the system name space does not need to be traversed, and you can use an unqualified lookup name. Note that this name does not resolve on other name servers. If an unqualified name is provided, the object key must be NameServiceServerRoot so that the correct initial context is selected. If a qualified name is provided, you can use the default key of NameService.

The following example shows a lookup of an EJB home. The enterprise bean that is being looked up is bound on the name server running on the host `myHost` on port `2809`. Note the object key of `NameServiceServerRoot`.

```
// Assume orb is an existing ORB instance
...
// Look up the home interface using the name under which the EJB home is bound
org.omg.CORBA.Object ejbHome = orb.string_to_object(
    "corbaname:iiop:myHost:2809/NameServiceServerRoot#mycompany/accounting");
accountHome =
    (AccountHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, AccountHome.class);
```

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:

Adjust timeout settings to improve handling of network failures. See “Object Request Broker service settings” for more information.

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.

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 to 300

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

Connection cache maximum:

Specifies the largest number of supported connections that can occupy the connection cache for the service. If simultaneous clients connect to the server-side ORB, this parameter can be increased up to 1000 clients to support the heavy load.

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	0-255

Connection cache minimum:

Specifies the smallest number of connections to be kept in the connection cache for the ORB service.

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	0-255

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.

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.

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 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.

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.

CAUTION:
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 EJB Specification, Version 2.0 (see section 5.4). Object references passed to 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

Use the this page to set and monitor settings associated with the Java Object Request Broker (ORB) service that do not appear on the main settings page by default.

Setting ORB properties through the administrative console

1. In the administrative console, click **Servers >Application servers >** *server_name* **>Container services > ORB service >Custom properties** .
2. To add properties to the page, click **New** and enter at least a name (case-sensitive) and a value for the property. Then click **Apply**.
3. When you are finished entering properties, click **OK**.

Setting ORB Properties through the command line

If you use the `java` command, use the `-D` option, for example:

```
java -Dcom.ibm.CORBA.propname1=value1 -Dcom.ibm.CORBA.propname2=value2 ... application name
```

If you use the `launchclient` command, 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 the product setup. A list of additional properties associated with the Java ORB service follows:

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 1298.

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 1298.

Default 2809

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. The setting of this property is valid only for client-side ORBs. Otherwise, the ORB obtains a value at run time by calling InetAddress.getLocalHost().getHostAddress() method.

com.ibm.CORBA.ServerSocketQueueDepth:

Corresponds to the length of a TCP/IP stack listen queue and prevents WebSphere Application Server from rejecting requests when space is not available in the listen queue. If several simultaneous clients connect to the server-side ORB, you can increase this parameter to support up to 1000 clients.

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.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: Do not configure logical pool distribution unless you have already configured it with 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 Logical pool distribution.

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.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Units	Milliseconds
Default	30
Range	20,000 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.

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.

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.

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.

LPD must be enabled (see com.ibm.websphere.threadpool.strategy.implementation).

Data type	Integer
Default	96
Range	10 minimum

com.ibm.CORBA.numJNIReaders:

You can improve performance by setting the com.ibm.CORBA.numJNIReaders system property through a command-line script. This property specifies the number of threads to share for request handling when the native selector mechanism is enabled.

Valid Range	0-2147483647
Default	2

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.websphere.ObjectIDVersionCompatibility:

This property applies when you have a mixed release cluster that has V6 and V5.1.0 or earlier and you are performing an incremental cell upgrade.

In mixed release cells, the migration program sets this property to 1.

After all of the cluster members are upgraded to V6, you can improve performance by removing this property or by setting the value to 2.

Setting the value to 1 indicates that the ORB runs using version 1 object identities, which is required to for mixed cells that contain application servers with releases prior to V5.1.1. In V6, not setting the property or changing the property value to 2 causes the ORB to run using version 2 object identities. Changing to version 2 object identities results in improved performance.

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 more information about the two properties and the order of precedence that the ORB uses to locate services, read the CORBA/IIOP specification, cited in “Object Request Brokers: Resources for learning” on page 1301.

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 (split here for publication only):

```
java program -ORBInitRef
     NameService=corbaloc::sample.server.com:2809/NameService
```

After these properties are set for services supported by the ORB, Java 2 Platform, Enterprise Edition (J2EE) applications obtain the initial reference to a given service by calling the resolve_initial_references function on the ORB, as defined in the CORBA/IIOP specification.

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 *installation_root*/java/ibm_bin directory , where *installation_root* is the installation directory for the product. 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 *installation_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 *installation_root*/java/ibm_bin directory instead of the JRE standard *installation_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 *installation_root*/java/ibm_lib directory.

Before using either the rmic or idlj tool, ensure that the *installation_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 *installation_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`. (In contrast, the default value for versions 3.5.x and 4.0.x of this product was `ISO8859_1`, also known as Latin-1.) 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`.

Note: If you are using a distributed application server with WebSphere Application Server for z/OS, you must set this property on the distributed client to `UCS2` or you might experience an exception.

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	

Java name	OSF registry ID	Comments
Cp1363C	0x10020553	
Cp1363	0x10020553	
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

For more information, read the CORBA and IIOP specification, cited in “Object Request Brokers: Resources for learning”

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” on page 1302
- “Programming specifications” on page 1302

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.

ORB services advanced settings on the z/OS platform

Use this topic to support Object Request Broker (ORB) service advanced settings. This support includes ORB listener keep alive, ORB Secure Sockets Layer (SSL) listener keep alive, control threads, and workload profile.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* > **Container services > ORB service** > **z/OS additional settings**.

ORB listener keep alive:

Defines the value in seconds provided to TCP/IP on the SOCK_TCP_KEEPALIVE option for the Internet Inter-ORB Protocol (IIOP) listener.

This option verifies that idle sessions are still valid by polling the client TCP/IP stack. If the client goes away without notifying the server, the session is still active on the server side. Use this property to clean up these unnecessary sessions. If the client does not respond, the session closes. The default is `0` (zero). If the property is not set, the TCP/IP option is not set. Setting the SOCK_TCP_KEEPALIVE option generates network traffic on idle sessions, which can cause problems.

Data type	Integer
Range	0 - 2147460

ORB SSL listener keep alive:

This property defines the value in seconds provided to TCP/IP on the SOCK_TCP_KEEPALIVE option for the SSL IIOP listener.

This option verifies if idle sessions are still valid by polling the client TCP/IP stack. If the client goes away without notifying the server, the session is still active on the server side. Use this option to clean up these unnecessary sessions. If the client does not respond, the session closes. The default is `0` (zero). If the property is not set, the TCP/IP option is not set. Setting the SOCK_TCP_KEEPALIVE option generates network traffic on idle sessions, which can be undesirable.

Data type	Integer
Range	0 - 2147460

Workload manager timeout:

Specifies the maximum time in seconds that IIOP requests are queued and dispatched to a servant process.

Data type	Integer
Range	0 - 2147040
Default	300
Disable workload manager queue timeout	0

Workload profile:

Specifies the server workload profile, which can be ISOLATE, IOBOUND, CPUBOUND, or LONGWAIT

The workload profile controls workload-pertinent decisions that are made by the WebSphere Application Server for z/OS run time, such as the number of threads used in the servant. The default value is IOBOUND, which is the appropriate value for most applications. Use one of the other values when your application requires more threads.

Workload profile	Number of Threads	Description
ISOLATE	1	Specifies that the servants are restricted to a single application thread. Use ISOLATE to ensure that concurrently dispatched applications do not run in the same servant. Two requests processed in the same servant can cause one request to corrupt another.
IOBOUND	MIN(30, MAX(5,(Number of CPUs*3)))	Specifies more threads in applications that perform I/O-intensive processing on the z/OS operating system. The calculation of the thread number is based on the number of CPUs. IOBOUND is used by most applications that have a balance of CPU intensive and remote operation calls. A gateway or protocol converter are two examples of applications that use the IOBOUND profile.
CPUBOUND	MAX((Number of CPUs-1),3)	Specifies that the application performs processor-intensive operations on the z/OS operating system, and therefore would not benefit from more threads than the number of CPUs. The calculation of the thread number is based on the number of CPUs. Use the CPUBOUND profile setting in CPU intensive applications, like XML parsing and XML document construction, where the vast majority of the application response time is spent using the CPU.
LONGWAIT	40	Specifies more threads than IOBOUND for application processing. LONGWAIT spends most of its time waiting for network or remote operations to complete. Use this setting when the application makes frequent calls to another application system, like Customer Information Control System (CICS) screen scraper applications, but does not do much of its own processing.

Note: Number of CPUs is the number of CPUs online when the controller comes up.

You can check the number of worker threads using message BBOO0234I in the servant job log. See WebSphere Application Server for z/OS Messages and codes for more information.

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 that 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. In addition, WebSphere Application Server for z/OS supports the coordination of resource managers through RRS (z/OS resource recovery services).
- 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

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 (for example, application servers) or Web Service Atomic Transaction (WS-AtomicTransaction) protocol. WebSphere Application Server also participates in transactions imported through J2EE Connector 1.5 resource adapters. WebSphere applications can also be configured interact with (or to direct the WebSphere transaction service to interact with) databases, JMS queues, and JCA connectors through their local transaction support when distributed transaction coordination is not required.

In addition to supporting the coordination of XAResource-based resource managers, WebSphere Application Server for z/OS supports the coordination of resource managers through RRS (z/OS resource recovery services). RRS-compliant resource managers include DB2, WebSphere MQ, IMS, and CICS. IBM WebSphere Application Server for z/OS is capable of coordinating a mix of RRSTransactional resource managers and XA capable resource managers under the same global transaction.

Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface or by supporting RRS) 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.

You can use transaction classes to classify client workload for workload management. The workload is different WebSphere transactions targeted to separate servant regions, each with goals defined by appropriate service classes. Each transaction is dispatched in its own WLM enclave in a servant region process, and is managed according to the goals of its service class. The server controller, which workload management views as a queue manager, uses the enclave associated with a client request to manage the priority of the work. If the work has a high priority, workload management can direct the work to a high-priority servant in the server. If the work has a low priority, workload management can direct the work to a low-priority servant. The effect is to partition the work according to priority within the same server.

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. The only exceptions to this behavior is when an application component dispatch occurs without container interposition; for example, for a stateless session bean `create`.

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.

Additionally, some JDBC Providers such as the DB2 for z/OS Local JDBC Provider support the use of z/OS Resource Recovery Service (RRS) to coordinate transaction processing. This type of JDBC Provider is RRSTransactional. When RRS is used, both local and global transactions are supported.

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 or RRS 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, see Editing deployment descriptors.

Peer recovery of transactions:

This topic describes peer recovery, which enables any server in a cluster to recover the transactional work for any other server in the same cluster.

This is a new feature in WebSphere Application Server version 6, and is in addition to the support for Peer restart and recovery, which enables you to restart on a peer system in the sysplex. The support for peer restart and recovery will be removed in a future version of WebSphere Application Server in favor of the peer recovery described in this topic.

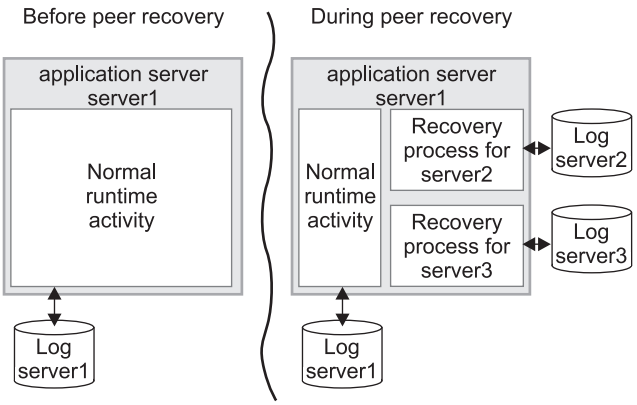
As a vital part of providing recovery for transactions, the transaction service logs information about active transactional work, such that the information is preserved across a server crash. This means that any transactional work in progress at the time of a server failure can be resolved when the server is restarted.

The standard recovery process performed when an application server restarts is for the server to retrieve and process the logged transaction information, to recover transactional work and complete in-doubt transactions. Completion of the transactional work (and hence the release of any database locks held by the transactions) is delayed until the server has successfully restarted and processed its transaction logs. If the server is slow to recover or requires manual intervention, the transactional work cannot be completed and access to associated databases is disrupted.

To minimize such disruption to transactional work and the associated databases, WebSphere Application Server provides a high availability strategy known as transaction peer recovery.

Peer recovery is provided within a server cluster. Each server in the cluster has a recovery process that can run alongside normal server activity, and enables a server in the cluster to recover the transactional

work for another server in the same cluster. There is no need to start a new application server specifically to recover the failed server.



The peer recovery process is the logical equivalent to restarting the failed server, but does not constitute a complete restart of the failed server within the peer server. It merely provides an opportunity for outstanding work to be completed. It is not possible for the peer recovery process to start new work beyond recovery processing. In other words, no “forward processing” is possible for the failed server. Both transactions and the compensation service fail over together to the same peer server.

Peer recovery moves the high availability requirements away from individual servers and onto the server cluster. After such failures, the WLM system of the cluster dispatches new work onto the remaining servers, the only difference from the users perspective being the potential drop in overall system throughput. If a server fails, all that is required is to tidy up work that was active on the failed server and redirect requests to an alternate server. Both transactions and the compensation service fail over together to the same peer server.

Common configuration for peer recovery

The transaction service requires a common configuration in order to be able to perform peer recovery between servers. This means that peer recovery processing can only take place between members of the same server cluster. Although a cluster can contain both version 5 and version 6 servers, peer recovery can only be performed between servers in the cluster that are at version 6 or later.

Control over which server is nominated to perform recovery processing for a failed peer is handled by the selected Clustered TM Policy of the cluster’s core group. The default “1 of N with preferred server” policy nominates a running member of the cluster to perform peer recovery processing and passes recovery control back to the failed server when it restarts.

By default, peer recovery is disabled until the **Enable high availability for persistent services** check box in the cluster configuration is selected. When this option has been selected, cluster members must be restarted before they engage in peer recovery processing for other cluster members. Similarly, if this option is disabled, cluster members must be restarted to prevent them from performing peer recovery.

For more information about high availability and core groups, see High availability groups and Core groups.

Location of recovery log files

The storage mechanism used to host recovery log files (for example, you can use IBM NAS and shared SCSI drives, but not simple network share) and access to that mechanism (for example, through a LAN), must support the file-based force operation that is used by the recovery log service to force data to disk. After the force operation is complete, information must be persistently stored on physical disk media; for example, IBM NAS (http://www.ibm.com/servers/storage/nas/index.html).

Interactions between the HA framework and the recovery log service must prevent concurrent access to a single physical recovery log.

Recovery log directory administration and scripting

You can configure the location of the transaction log directory using either the WebSphere administrative console or commands. For peer recovery, the configuration is stored as part of the recovery log configuration in the serverindex.xml node-level configuration file.

To ease migration of the transaction log configuration from previous versions of WebSphere Application Server, special logic has been added to the administrative console. This is to help migration of the transaction log directory configuration from the original server.xml server-level configuration file to the serverindex.xml node-level configuration file.

- Changes to recovery log directory settings are always stored within the new serverindex.xml file.
- Scripted modifications that configure the original recovery log settings, or migration of version 5 application servers to version 6, cause the original transaction log directory configuration to be updated. The administrative console detects this condition and prompts the user to save the configuration when they view the transaction service panel. This save operation saves the changed configuration to the serverindex.xml file, and resets the older fields to null.
- New scripting should target the serverindex.xml configuration directly. Existing scripting that targets the server.xml configuration should be changed to target the serverindex.xml at the earliest opportunity.

Peer recovery example

The following diagrams illustrates the peer recovery process that takes place if a single server fails. Figure 1 shows three stable servers running in a WAS cluster. The WLM engine is load balancing work between these servers which results in locks being held by the backend database on behalf of each of them. In addition, communication has taken place between servers 1 and 2 which now retain references to each other.

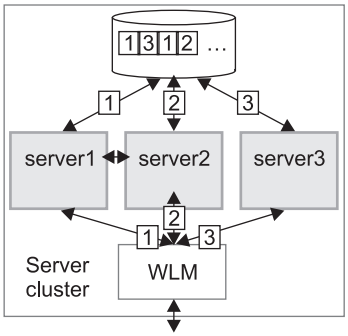


Figure 12. Server cluster up and running, just before server failure

Figure 2 shows the state of the system after server 1 has failed without clearing locks from database. Servers 2 and 3 were able to run their existing transactions to completion and release existing locks in the backend database, but further access may be impaired because of the locks still held on behalf of server 1. In practice, some level of access by servers 2 and 3 should still be possible, assuming appropriately configured lock granularity, but for this example assume that servers 2 and 3 have attempted to access locked records and become blocked.

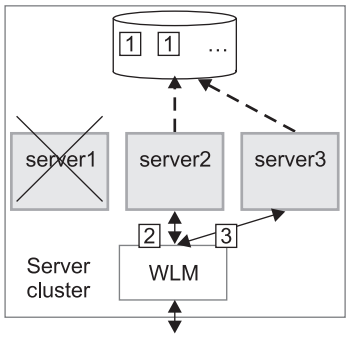


Figure 13. Figure 2 - Server 1 fails. Servers 2 and 3 become blocked as a result.

Figure 3 shows a peer recovery process for server 1 running inside server 3. The transaction service portion of the recovery process retrieves the information persisted by server 1, and uses that information to complete any in-doubt transactions. In addition, the cluster redirects endpoint references for server 1 to server 3. In this figure, the peer recovery process is partially complete as some locks are still held by the database on behalf of server 1.

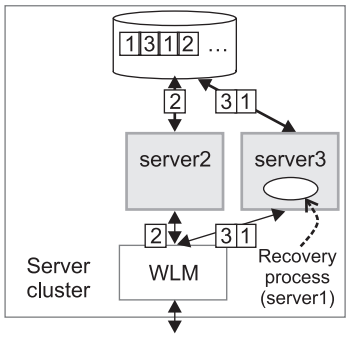


Figure 14. Figure 3 - Peer recovery process started in server 3

Figure 4 shows the state of the server cluster when the peer recovery process has completed. The system is in a stable state with just two servers, between which the WLM engine can balance workload. Server 1 can be restarted at some time in the future, when it will have no recovery processing of its own to perform.

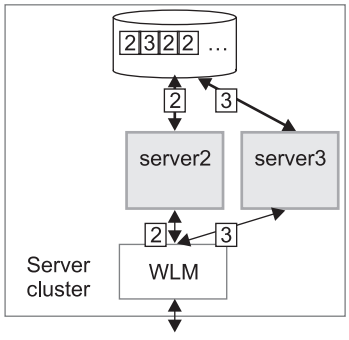


Figure 15. Figure 4 - Server cluster stable again with just two servers - server 2 and server 3.

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 WTRN0105I 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.

Web Services – Atomic Transaction for WebSphere Application Server:

The Web Services - Atomic Transaction for 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 intreface. 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 16, shows a transaction context shared between two WebSphere application servers for a Web service request that contains a WS-AT CoordinationContext.

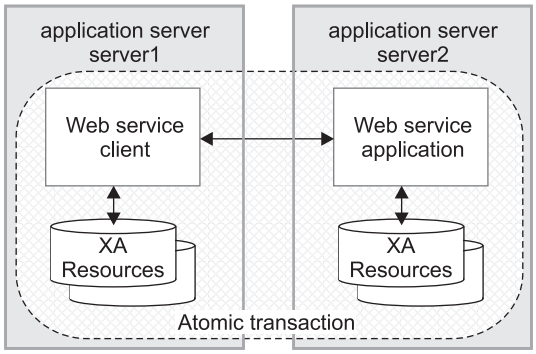


Figure 16. Transaction context shared between two WebSphere application servers.

WS-AT support restrictions

In WebSphere Application Server version 6.0, WS-AT contexts cannot be propagated through firewalls and cannot be started from a non-recoverable client process.

Work requests for the same WS-AT transaction sent to a server cluster are not guaranteed to be assigned to the same cluster member every time. In such cases, the work for a transaction might be handled by multiple cluster members. If transactional work of multiple cluster members contends over the same transactional resource then a deadlock condition can result.

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 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.

Using 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.

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 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 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 use a single non-XA resource and one or more XAResources.

  Use the Last Participant Support.

Points specific to WebSphere Application Server for z/OS

- You want to use a non-XA resource along with multiple two-phase RRS resources.

  A non-XA resource in a transaction along with RRS resources is supported any time a global transaction is active. A global transaction is active when the deployment descriptor has **Transaction** set to `Supports`, `Required`, `Requires New`, or `Mandatory`. Global transactions also are active for component-managed deployments.

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.

Perhaps you 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:** 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.
6. **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.

7. Review or change other configuration properties, to suit your requirements. For more information about the properties of the transaction service, see “Transaction service settings.”
8. Click **OK** and save.
9. 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 modify transaction service settings.

To view this administrative console page, click **Servers > Application Servers >** *server_name* > **Container Services > Transaction Service**.

Transaction log directory:

Specifies the location of the JTA Partner Log.

This change is applicable only to the configuration where the application uses distributed resources or XA transactions, for example, multiple databases and resources are accessed within a single transaction.

On z/OS, this log is used for recovery of XA resources. When the application that runs on the WebSphere product accesses XA resources, the WebSphere product stores information about the resource to enable XA transaction recovery.

Syntax

[*location type URL tag*] *location specification*

where

- *location type URL tag* specifies the optional location type for the JTA Partner Log:
  - dir:// specifies that the JTA Partner Log location is in a fully qualified HFS directory specified by *location specification*. dir:// is the default.
- *location specification* specifies the location name for the JTA Partner Log:
  - To specify a logstream, use the syntax `logstream://HLQ`
  - If the *location type URL tag* is dir://, use a fully qualified HFS directory for the *location specification*. The complete name of the directory must be unique within the WebSphere node.

Default

dir://*install root*/tranlog/*server name*

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 maximum duration, in seconds, for transactions on this application server.

Component-managed transactions that do not have a timeout explicitly set are also assigned this value.

Any transaction that is not requested to complete before this timeout is rolled back. If set to 0, only the maximum transaction timeout configuration value applies.

Data type	Integer
Units	Seconds
Default	120
Range	0 to 2 147 040

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 rolling back in this application server. If set 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 duration, in seconds, that transactions started by or propagated into this application server are allowed to run.

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 2 147 040

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.

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:

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

If you leave this value at 0, the application server still makes subsequent attempts to complete the transaction. It incrementally lengthens the wait time before each attempt, to improve system throughput.

Heuristic completion direction:

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.

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.

Manual transactions:

The number of transactions awaiting 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:

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:

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:

The number of transactions 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**.

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.

WebSphere Application Server for z/OS can relegate the resource manager to an in-doubt (prepared) state *only if* the RRS transaction context is available.

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.)

WebSphere Application Server for z/OS can resolve the transactions according to the selections made on this page *only if* the RRS transaction context is available.

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_typelocal_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.

Configuring transaction properties for peer recovery

Use this task to configure the transaction properties required for peer recovery of failed application servers in a cluster.

Peer recovery of transactions is a new feature in WebSphere Application Server version 6, and is in addition to the support for Peer restart and recovery, which enables you to restart on a peer system in the sysplex. The support for peer restart and recovery will be removed in a future version of WebSphere Application Server in favor of the peer recovery described in this topic. For more information about configuring peer restart and recovery, see Setting up peer restart and recovery.

The transaction service requires a common configuration in order to be able to perform peer recovery between servers. This means that peer recovery processing can only take place between members of the same server cluster. Although a cluster can contain both version 5 and version 6 servers, high availability for peer recovery must only be enabled and configured if all servers in the cluster are at version 6.

Configuring the transaction properties required for peer recovery is part of the overall task for configuring a cluster to use high availability support.

To configure the transaction properties required for peer recovery, complete the following steps:

1. Configure the transaction log directory setting for each server in the cluster. You can configure the location of the transaction log directory using either the WebSphere administrative console or commands.

For peer recovery, each server in the cluster must be able to access the log directories of other servers in the same cluster. Interactions between the High Availability framework and the recovery log service prevent concurrent access to a single physical recovery log.

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.

The storage mechanism used to host recovery log files (for example, you can use IBM NAS and shared SCSI drives, but not simple network share) and access to that mechanism (for example, through a LAN), must support the file-based force operation that is used by the recovery log service to force data to disk.

For more information about configuring transaction log directories, see Configuring transaction properties for an application server.

2. Enable the High Availability function for the cluster, by completing the following steps on the cluster configuration panel of the WebSphere administrative console:
   a. In the administrative console, click **Servers** → **Clusters** → *cluster_name*.
   b. To enable high availability for a cluster, select the **Enable high availability for persistent services** option.
   c. To change the peer recovery characteristics for a specific server in the cluster, you need to create a new policy that has match criteria for the specific server.

      For more information about creating a new policy, see Creating a policy for a high availability group.

   For more information about enabling the High Availability function for a cluster, see Server cluster settings
3. Change the WAS_TRANSACTION policy in the core group configuration
   a. In the administrative console, click **Servers** → **Core Groups** → **DefaultCoreGroup** → **Policies** → **Cluster TM Policy**.
   b. Change the policy properties to suit your recovery requirements. For example, you can enable or disable the use of hardware quorum support.

   For more information about configuring the WAS_TRANSACTION policy in the core group configuration, see Core groups.

Managing active and prepared transactions

Use this task to manage active and prepared transactions that might need administrator action.

Under normal circumstances, every effort is made to finish a transaction. However, due to RRS and native contexts finishing, finishing the transaction may not be possible. In this case, the transaction is marked rollback_only so that it rolls back at the next available window. In other situations, you may need to finish a transaction manually. For example, you may want to finish 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.

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 (for example, application servers) 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.

Using one-phase and two-phase commit resources in the same transaction

Use these topics to help you coordinate the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction.

You can coordinate the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction.

At transaction commit, the two-phase commit resources are prepared first using the two-phase commit protocol, and if this is successful the one-phase commit-resource is then called to commit(one_phase). The two-phase commit resources are then committed or rolled back depending on the response of the one-phase commit resource.

For more information about using one-phase and two-phase commit resources within the same transaction, see the following topics:
- Coordinating use of one-phase and two-phase commit resources within the same transaction
- Assembling an application to use one-phase and two-phase commit resources in the same transaction
- Configuring an application server to allow logging for heuristic reporting

Coordinating access to 1-PC and 2-PC-capable resources within the same transaction

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.

At transaction commit, the two-phase commit resources are prepared first using the two-phase commit protocol, and if this is successful the one-phase commit-resource is then called to commit(one_phase). The two-phase commit resources are then committed or rolled back depending on the response of the one-phase commit resource.

Note: If the global transaction is distributed across multiple application servers *that are all running at WebSphere Application Server version 5.1 or later*, you can coordinate access to one-phase and two-phase commit capable resources within the same transaction.

Note: If the global transaction is distributed across multiple application servers *that are all running at WebSphere Application Server version 5.1 or later* then you can exploit last participant support to coordinate a one-phase commit capable resource and any number of two-phase commit capable resources within the same transaction, in a limited number of scenarios.

- The main scenario is where the one-phase commit resource provider is accessed in the application server process (the “transaction root” server) in which the transaction is started.

  In this scenario, last participant support can coordinate a one-phase commit capable resource and any number of two-phase commit capable resources within the same transaction.
- If the one-phase commit resource provider is accessed in a different application server (a “transaction subordinate” server) from the one in which the transaction was started; for example, as a result of a transactional invocation on a remote EJB interface where the EJB implementation accesses a one-phase commit resource provider.

  In this scenario, the transaction typically cannot be committed. To be able to commit (as part of a global transaction) a one-phase commit resource enlisted on a transaction subordinate server, the transaction service must delegate coordination responsibility from the transaction root to the subordinate server. This occurs only if no other resources were registered with the transaction root server.

Last participant support introduces an increased risk of an heuristic outcome to the transaction. That is, the transaction manager cannot be sure that all resources were completed in the same direction (either

committed or rolled back). For this reason, to enable an application to coordinate access to one-phase and two-phase commit capable resources within the same transaction, you configure the application to accept the increased risk of an heuristic outcome.

An heuristic outcome occurs if the transaction service (JTS) receives no response from the commit one-phase flow on the one-phase commit resource. In this situation the transaction service cannot determine whether changes for the one-phase commit resource were committed or rolled back, so cannot drive reliably the correct outcome of the global transaction on the other two-phase commit resources.

You can configure the transaction service for an application server to indicate whether or not to log that it is about to commit the one-phase commit resource. This does not reduce the heuristic hazard, but ensures that any failure, and subsequent recovery, of the application server during the one-phase commit phase occurs with knowledge of whether or not the one-phase commit resource was asked to commit:

- If the one-phase commit resource was asked to commit, a heuristic outcome is reported to the activity log.
- If the one-phase commit resource was not asked to commit, then the transaction is rolled back consistently.

Assembling an application to use one-phase and two-phase commit resources in the same transaction

Use this task to assemble an application to use one-phase and two-phase commit resources within the same transaction.

To enable an application to use one-phase and two-phase commit capable resources within the same transaction, you must configure the deployment attributes of the application to accept the increased risk of an heuristic outcome.

You can configure the deployment attributes of an application by using an assembly tool such as the Application Server Toolkit (AST) or Rational Web Developer.

This task description assumes that you have an EAR file for an application component, that can be deployed in WebSphere Application Server. For more details about assembling applications, see Assembling applications.

To configure an application to indicate that you accept the increased risk of an heuristic outcome, complete the following steps:

1. Start the assembly tool.
2. Create or edit the application EAR file.

   Note: Ensure that you set the target server as WebSphere Application Server version 6.

   For example, to change attributes of an existing application, use the import wizard to import the EAR file into the assembly tool. To start the import wizard:

   a. Click **File-> Import-> EAR file**
   b. Click **Next**, then select the EAR file.
   c. In the Target server field, select `WebSphere Application Server v6.0`
   d. Click **Finish**
3. In the J2EE Hierarchy view, right-click the Enterprise Application instance, then click **Open With > Deployment Descriptor Editor**. A property dialog notebook for the component is displayed in the property pane.
4. In the property pane, select the Extended Services tab.
5. In the Last Participant Support section, select the **Last participant support** checkbox.
6. Save your changes to the deployment descriptor.
   a. Close the Deployment Descriptor Editor.

b. When prompted, click **Yes** to indicate that you want to save changes to the deployment descriptor.

7. Verify the archive files.
8. From the popup menu of the project, click **Deploy** to generate EJB deployment code.
9. **Optional:** Test your completed module on a WebSphere Application Server installation. Right-click a module, click **Run on Server**, and follow the instructions in the displayed wizard. Note that **Run on Server** works on the Windows, Linux/Intel, and AIX operating systems only; you cannot deploy remotely from the Application Server Toolkit (AST) or Rational Web Developer to a WebSphere Application Server installation on a UNIX operating system such as Solaris.

   Important

   Important: Use **Run On Server** for unit testing only. The Application Server Toolkit (AST) or Rational Web Developer controls the WebSphere Application Server installation and, when an application is published remotely, the assembly tool overwrites the server configuration file for that server. Do not use on production servers.

After assembling your application, use a systems management tool to deploy the EAR file onto the application server that is to run the application; for example, using the administrative console as described in Deploying and managing applications.

Related tasks

Assembling applications
Application assembly consists of creating Java 2 Platform, Enterprise Edition (J2EE) modules that can be deployed onto application servers. The modules are created from code artifacts such as Web application archives (WAR files), resource adapter archives (RAR files), enterprise bean (EJB) JAR files, and application client archives (JAR files). This packaging and configuring of code artifacts into enterprise application modules (EAR files) or standalone Web modules is necessary for deploying the modules onto an application server.

Last participant support extension settings:

Use this page to configure last participant support extensions.

Last participant support is an extension to the transaction service to allow a single one-phase resource to participate in a two-phase transaction with one or more two-phase resources.

To view this administrative console page, click **Applications** *application_name* → **Last Participant Support Extension**

Related tasks

“Assembling an application to use one-phase and two-phase commit resources in the same transaction” on page 1327
Use this task to assemble an application to use one-phase and two-phase commit resources within the same transaction.

Accept Heuristic Hazard:

Specifies whether the application accepts the possibility of an heuristic hazard occurring in a two-phase transaction containing a one-phase resource.

Default Cleared

Range

Selected
The application accepts the increased risk of an heuristic outcome.

Cleared
The application does not accept the increased risk of an heuristic outcome.

Configuring an application server to log heuristic reporting

To enable an 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, use the Administrative console to 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. Select the Transaction Service tab, 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. Select the Configuration tab, to display the transaction-related configuration properties.
5. Select the **Enable logging for heuristic reporting** checkbox.
6. Click **OK**.
7. Stop then restart the application server.

Exceptions thrown for transactions involving both single- and two-phase commit resources

The exceptions that can be thrown by transactions that involve single- and two-phase commit resources are the same as those that can be thrown by transactions involving only two-phase commit resources.

The exceptions that can be thrown are listed in the WebSphere Application Server application programming interface reference information (Javadoc).

Last Participant Support: Resources for learning

Use the links in this topic to find relevant supplemental information about Last Participant Support. 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.

View links to additional information about:
- “Programming specifications”
- “Other”

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

Other
- WebSphere Application Server Enterprise Version 5 Overview: Advanced Transactional Connectivity
- Listing of PDF files to learn about WebSphere Application Server Version 5
- Listing of all IBM WebSphere Application Server Redbooks
- Listing of all IBM WebSphere Application Server Whitepapers
- WebSphere Application Server Enterprise Edition 4.0: A Programmer’s Guide

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 an introduction to each WebSphere extension.

ActivitySessions	How do I?...	Overview	Samples
Application profiling	How do I?...	Overview	Samples
Asynchronous beans	How do I?...	Overview	Samples
Dynamic caching	How do I?...	Overview	
Dynamic query	How do I?...	Overview	Samples
Internationalization	How do I?...	Overview	Samples
Object pools	How do I?...	Overview	
Scheduler	How do I?...	Overview	Samples
Startup beans	How do I?...	Overview	
Work areas	How do I?...	Overview	

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. 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
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.

ActivitySession service settings:

Use this page to administer the runtime properties of the ActivitySession service.

To view this administrative console page, click **Servers** → **Application servers** → *server_name* → **[Container Settings] Business Process Services** → **Activity session 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:

The default timeout for an ActivitySession. A server resets an ActivitySession if a remote client has failed to complete the ActivitySession within this time period.

The default ActivitySession timeout specifies the time after which an ActivitySession is completed automatically by the ActivitySession service, if a remote client has failed to complete the ActivitySession within the specified time. 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

Using the ActivitySession service

These topics provide information about implementing WebSphere enterprise applications that use ActivitySessions.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. ActivitySessions provide a scoping mechanism for units of work, and both an ActivitySession and a transaction has the same following characteristics:
- It can be bean-managed or container-managed
- It can be distributed across application servers

- It can be used as the context for managing EJB activation policy and lifecycle

An ActivitySession differs significantly from a transaction in the manner of its interaction with resource managers. An ActivitySession is used to scope or coordinate local transactions. That is, an ActivitySession can be used to request multiple one-phase resource managers to come to an application- or container-determined outcome. Unlike a transaction, an ActivitySession has no notion of a prepare phase or any notion of recovery at a service level.

The WebSphere EJB container and deployment tools support ActivitySessions as an extension to the J2EE programming model. Enterprise beans can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An enterprise bean with an ActivitySession-scoped lifecycle can participate in a resource manager local transaction (RMLT) that has a duration of the ActivitySession rather than an individual method on the bean (which is all that is possible under the standard J2EE model). Applications can then be composed of several enterprise beans with ActivitySession-based activation, with each bean participating in extended local transactions with one or more resource managers. At the end of the ActivitySession each of the local transactions can be directed to a common outcome by the ActivitySession manager.

You can configure the WebSphere containers and deployable applications to support enterprise beans that operate under application- or container-initiated ActivitySessions rather than, or in addition to, transactions.

For more information about implementing WebSphere enterprise applications that use ActivitySessions, see the following topics:

The ActivitySession service:

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

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

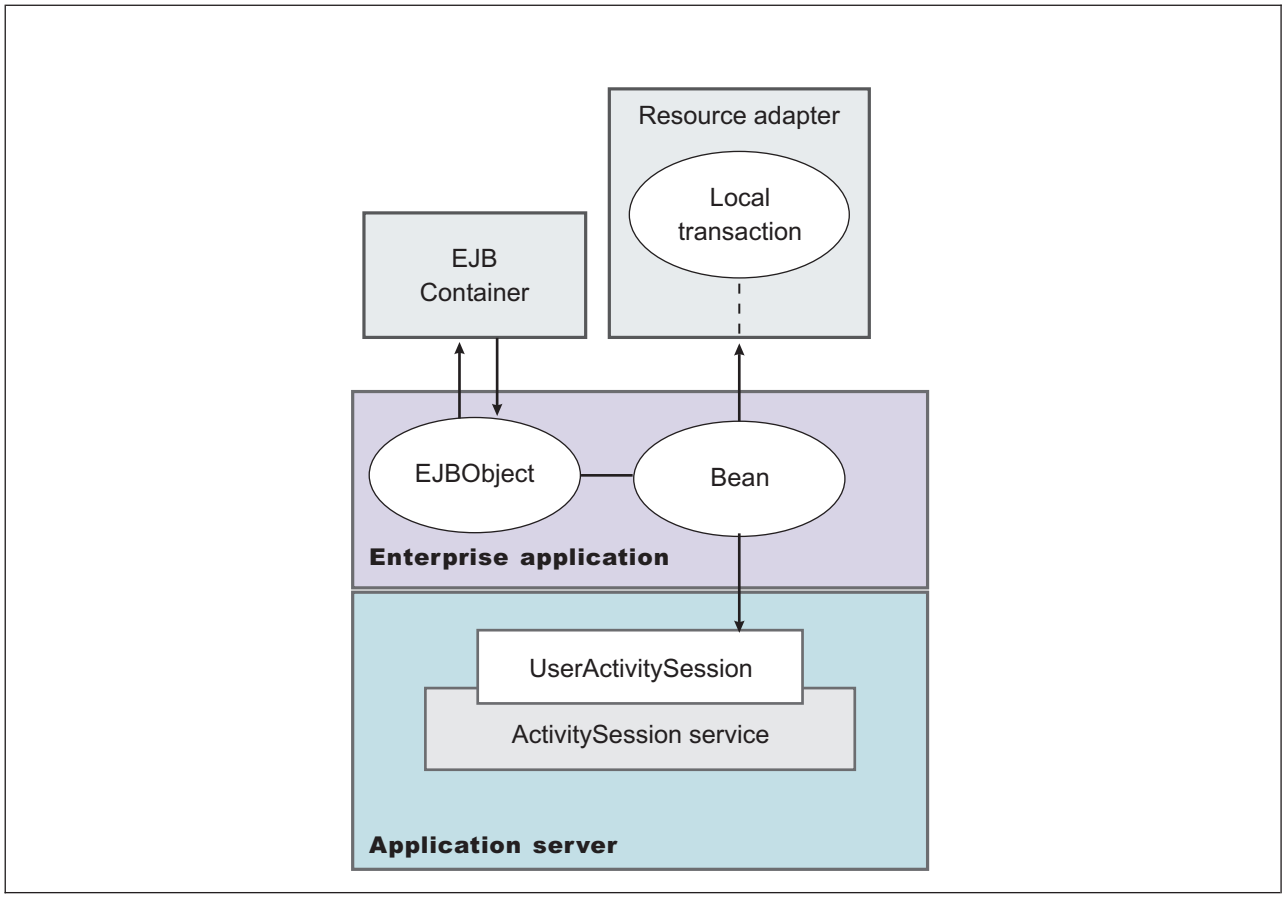


Figure 17. The ActivitySession service. This figure show the main components of the ActivitySession service within WebSphere Application server. For an overview of these components, see the text that accompanies this figure.

Although the purpose of a global transaction is to coordinate multiple resource managers, global transaction context is often used by J2EE applications as a “session” context through which to access EJB instances. An ActivitySession context is such a session context, and can be used in preference to a global transaction in cases where coordination of two-phase commit resource managers is not needed. Further, an ActivitySession can be associated with an HttpSession to extend a “client session” to an HTTP client.

ActivitySession support is available to Web, EJB, and J2EE-client components. EJB components can be divided into beans that exploit container-managed ActivitySessions and beans that use bean-managed ActivitySessions.

The ActivitySession service provides a UserActivitySession application programming interface available to J2EE components that use bean-managed ActivitySessions for application-managed demarcation of ActivitySession context. The ActivitySession service also provides a system programming interface for container-managed demarcation of ActivitySession context and for container-managed enlistment of one-phase resources (RMLTs) in such contexts.

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

A common scenario is a J2EE application accessing one or more enterprise beans backed by non-transactional (one-phase commit) resources. The application, or its container, uses the UserActivitySession interface to define the demarcation boundaries within which operations against the enterprise beans are grouped and to control whether those grouped operations should be checkpointed or

discarded. The business logic of the enterprise beans does not need to use any ActivitySession interfaces. The container into which the enterprise beans are deployed ensures that updates to the underlying one-phase resource managers are coordinated.

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

Related tasks

Developing a J2EE application to use an ActivitySession
This topic provides an overview of the scenarios for which you would develop a J2EE application to use an ActivitySession.

Using ActivitySessions with HTTP sessions:

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

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

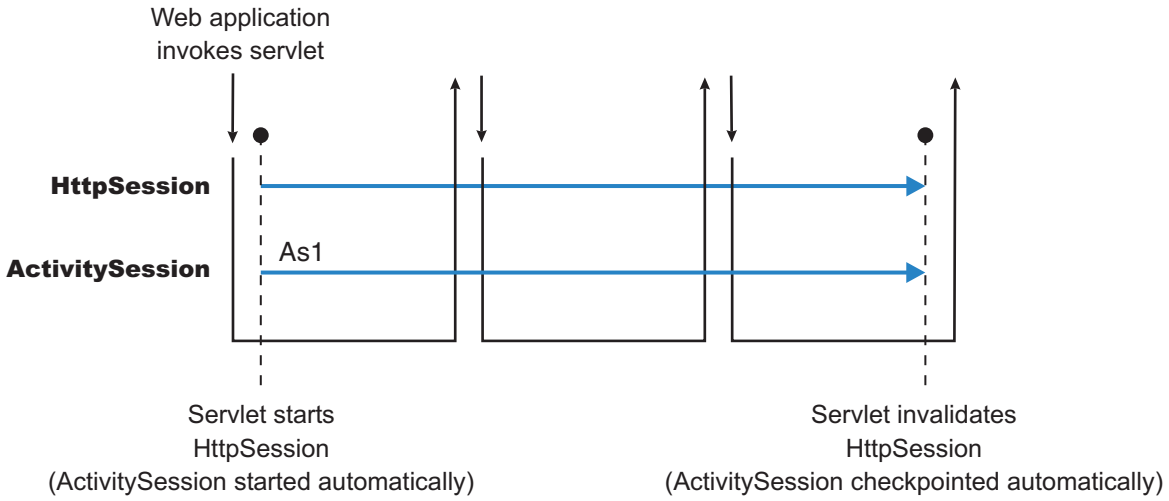
An ActivitySession is associated automatically with an HttpSession, so can be used to extend access to the ActivitySession over multiple HTTP invocations, over inclusion or forwarding of servlets, and to support EJB activation periods that can be determined by the lifecycle of the web HTTP client. An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

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

- The Web container starts and ends ActivitySessions.

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

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

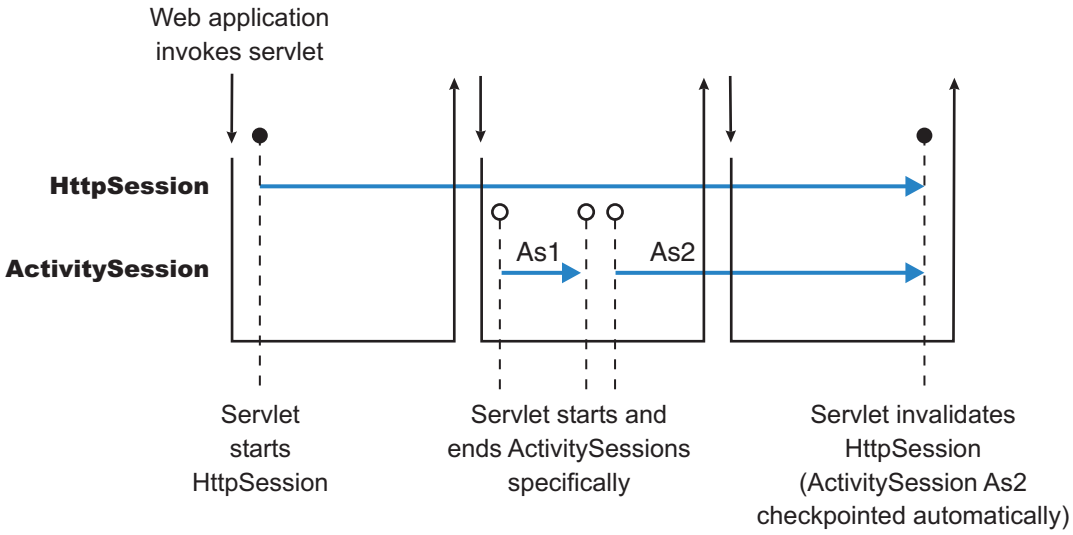


- The Web application starts and ends ActivitySessions.

  The Web application invokes a servlet that has been configured for application control of ActivitySesions.
  - If an HttpSession exists and has an associated ActivitySession, the servlet can use or end that ActivitySession context.
  - If an HttpSession does not exist, the servlet can start an HttpSession, but this does not automatically start an ActivitySession.
  - If an HttpSession exists but does not have an associated ActivitySession, the servlet can start a new ActivitySession. This automatically associates the ActivitySession with the HttpSession. The ActivitySession lasts either until the ActivitySession is specifically ended or until the HttpSession is ended.

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

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



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

The following points apply to both usage models:

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

Related concepts

The ActivitySession service programming interfaces
The ActivitySession service consists of an application programming interface available to Web applications, session EJBs, and J2EE client applications for application-managed demarcation of ActivitySession context.

Related tasks

Configuring Web module ActivitySession deployment attributes
Use this task to set the ActivitySession deployment attributes for a Web application to start UserActivitySessions and perform work scoped within ActivitySessions.

ActivitySession and transaction contexts:

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

An ActivitySession context is very similar to a transaction context and extends the lifecycle choices for activation of enterprise beans; it can encapsulate one or more transactions. The ActivitySession context is a distributed context that, like the transaction context, can be bean- or container-managed. An

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

ActivitySession context is used mainly by a client to scope the lifecycle of an enterprise bean that it uses either beyond or in the absence of individual transactions started by that client.

ActivitySessions have a lower overhead than transactions and can be used instead of transactions that are only used to scope the lifecycle of a called enterprise bean. For a bean with an activation policy of ActivitySession, the duration of any resource manager local transactions (RMLTs) started by that bean can be bounded by the duration of the ActivitySession instead of the bean method in which the RMLT was started. This provides flexibility and potential for using RMLTs in an enterprise bean beyond the scenarios described in the J2EE specifications. The J2EE specifications define that RMLTs need to be completed before the end of the bean method, because the bean method is the only containment boundary for local transactions available in those specifications.

The following rules defines the relationship between transactions and ActivitySessions.

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

Related concepts

The ActivitySession service programming interfaces
The ActivitySession service consists of an application programming interface available to Web applications, session EJBs, and J2EE client applications for application-managed demarcation of ActivitySession context.

Related reference

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

Combining transaction and ActivitySession container policies:

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

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

In each row, the final column describes the behavior that the EJB container takes with respect to global transaction and ActivitySession context, based on the following abbreviations:
S*n* An ActivitySession, where *n* indicates the ActivitySession instance.
T*n* A transaction, where *n* indicates the transaction instance.

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

The rows highlighted in bold are not allowed.

Table 16. Container behavior for activitysession and transaction policies deployment settings

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

Table 16. Container behavior for activitysession and transaction policies deployment settings (continued)

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

Table 16. Container behavior for activitysession and transaction policies deployment settings (continued)

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

Table 16. Container behavior for activitysession and transaction policies deployment settings (continued)

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

Table 16. Container behavior for activitysession and transaction policies deployment settings (continued)

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

Table 16. Container behavior for activitysession and transaction policies deployment settings (continued)

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

Related concepts

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

Related tasks

Configuring EJB module ActivitySession deployment attributes
Use this task to set the ActivitySession deployment attributes for an enterprise bean to enable the bean to participate in an ActivitySession context and support ActivitySession-based operations.

The ActivitySession service application programming interfaces:

The ActivitySession service consists of an application programming interface available to Web applications, session EJBs, and J2EE client applications for application-managed demarcation of ActivitySession context.

Applications use the UserActivitySession interface, which provides demarcation scope methods.

ActivitySession API

The ActivitySession service provides the UserActivitySession interface for use by EJB Session beans using bean-managed context demarcation, Web application components configured with **ActivitySession control**=`Web Application`, and J2EE client applications. This UserActivitySession interface defines the set of ActivitySession operations available to an application component. An implementation of this interface is obtained via a JNDI lookup of the URL ″java:comp/websphere/UserActivitySession″. It is used to begin and end ActivitySessions and to query various attributes of the active ActivitySession associated with the thread.

For more information about the ActivitySession API, see WebSphere Application Server application programming interface reference information (Javadoc).

The ActivitySession API and the implementation of its interfaces is contained in the com.ibm.websphere.ActivitySession package.

Programming Examples

The following code extract provides a basic example of using the UserActivitySession interface:

```
// Get initial context
  InitialContext ic = new InitialContext();
// Lookup UserActivitySession
  UserActivitySession uas = (UserActivitySession)ic.lookup("java:comp/websphere/UserActivitySession");

// Set the ActivitySession timeout to 60 seconds
  uas.setSessionTimeout(60);
// Start a new ActivitySession context
  uas.beginSession();
// Do some work under this context
  MyBeanA beanA.doSomething();
  ...
  MyBeanB beanB.doSomethingElse();
// End the context
  uas.endSession(EndModeCheckpoint);
```

Samples: ActivitySessions:

This topic describes the ActivitySession samples provided with WebSphere Application Server.

MasterMind sample

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

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

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

J2EE client container application and a CMP entity bean backed by a one-phase commit datasource

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

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

The client accesses the UserActivitySession, begins an ActivitySession, updates two instances of the bean, then ends the ActivitySession. It does this twice using EndModeReset then EndModeCheckpoint. This sample demonstrates the following functionality:
- Client access to the UserActivitySession interface
- Multiple RMLTs being scoped to the ActivitySession and automatically taking their completion direction from that of the ActivitySession

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

A J2EE client container application and two session beans with different ActivitySession types

This sample consists of a J2EE client container application and the following session beans:
- SLB1, a stateless session bean configured with an ActivitySession Type of Bean.
- SFB2, a stateful session bean configured with ActivitySession Type of Requires, an LTC boundary of ActivitySession, LTC Resolution Contol of APPLICATION, and an LTC Unresolved Action of ROLLBACK.

Both beans are configured with TX_NOTSUPPORTED.

This sample performs the following steps:
1. The client starts SLB1
2. SLB1 accesses the UserActivitySession interface, begins an ActivitySession, then calls a method on SFB2
3. SFB2 accesses the UserActivitySession interface, begins an ActivitySession, calls a method on SFB2
4. SFB2 gets a connection (setAutoCommit false) then uses JDBC to update a single-phase datasource.
5. SLB1 then optionally calls a seperate method on SFB2 to finish the work either committing or rolling-back the RMLT.
6. SLB1 then ends the ActivitySession with an EndModeCheckpoint.

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

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

ActivitySession service: Resources for learning:

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

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

View links to additional information about:

- “Programming model and decisions” on page 1347
- “Programming specifications” on page 1347

- “Other”

Programming model and decisions
- WebSphere Application Server application programming interface reference information (Javadoc)

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

Other
- WebSphere Application Server Enterprise Version 5 Overview: Advanced Transactional Connectivity
- Listing of PDF files to learn about WebSphere Application Server Version 5
- Listing of all IBM WebSphere Application Server Redbooks
- Listing of all IBM WebSphere Application Server Whitepapers
- WebSphere Application Server Enterprise Edition 4.0: A Programmer’s Guide

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.

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 1331
Use this page to administer the runtime properties of the ActivitySession service.

Application profiling

Task overview: Application profiling

Application profiling enables you 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 run time environment. For more information, see Application Profiling: Overview.

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: Overview: Application profiling enables you to identify particular units of work to the WebSphere Application Server run time environment. The run time can tailor its support to the exact requirements of that unit of work. Access intent is currently the only run time 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 6.0 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.

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 run time 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.

Related tasks

Assembling applications for application profiling

"Managing application profiles" on page 1355

"Using the TaskNameManager interface" on page 1356

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 6.0 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.

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.

Related concepts

"Application profiling tasks" on page 1353
Tasks are named units of work. They are the mechanism by which the run time environment determines which access intent policies to apply when an entity bean's data is loaded from the back end system.

Related tasks

Creating an application profile

"Managing application profiles" on page 1355

Automatically configuring application profiles and tasks

Application profiling performance considerations: Application profiling enables assembly configuration techniques that improve your application runtime, 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 run-time 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 runtime 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 run time 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 run time 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.

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 are initiated after the task name is set. Application managed tasks are configured on all J2EE components.

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.

Related concepts

"Application profiles" on page 1349
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*).

Related tasks

Automatically configuring application profiles and tasks

"Using the TaskNameManager interface" on page 1356

Configuring container managed tasks for application clients

Configuring container managed tasks for Web components

Configuring container managed tasks for Enterprise Java Beans

Configuring application managed tasks for application clients

Configuring application-managed tasks for Web components

Configuring application managed tasks for Enterprise JavaBeans

Application profiling interoperability:
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 if the Application Profiling 5.x Compatibility Mode attribute is turned on. If the 5.x Compatibility Mode attribute is off, Version 5 application profiles might display unexpected behavior. For information about the 5.x Compatibility Mode, see "Application profiling service settings" on page 1358.

Similarly, application profiles that you create using WebSphere Application Server Version 6 are not compatible with Version 5 or earlier versions. Even applications configured with application profiles run on

Version 6 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 6.0 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.

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.

If a clustered environment contains both Version 5.x and 6.0 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 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.

Related tasks

"Using the TaskNameManager interface" on page 1356

Related reference

"Application profiling service settings" on page 1358
Use this page to enable or disable the application profiling service.

Managing application profiles

Manage your application profiles using the administrative console. From the console, you can add tasks to, and 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 affect.

Related tasks

Assembling applications for application profiling

Related reference

"Task collection" on page 1360
Use this page to manage tasks.

Using the TaskNameManager interface:

You can declaratively configure container managed tasks for Java 2 platform, Enterprise Edition (J2EE) web components, application clients, and Enterprise JavaBeans (EJB). On rare occasions, you might find it necessary to *programmatically* set the current task name. Application profiling supports this requirement with a simple interface that 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 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 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.
}
// . . .
//
```

Resetting the task name has no effect unless called by a J2EE 1.3 application executing 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.

Related tasks

Configuring container managed tasks for Web components

Configuring container managed tasks for application clients

Configuring container managed tasks for Enterprise Java Beans

TaskNameManager interface: The TaskNameManager is the programmatic interface to the application profiling function. Application profiling enables you to identify particular units of work to the WebSphere Application Server run time environment. The run time can tailor its support to the exact requirements of that unit of work. Access intent is currently the only run time 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.

Programmers can declaratively configure container managed tasks for Java 2 platform, Enterprise Edition (J2EE) web components, application clients, and Enterprise JavaBeans (EJB). On rare occasions, it may be necessary to programmatically set the current task name. Application profiling supports this requirement with the TaskNameManager interface that 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 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**.

Related tasks

“Managing application profiles” on page 1355

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 6.0 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.

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 6.0. 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 the assembly tools such as Application Server Toolkit (AST) or Rational Web Developer to add or delete application profiles. The AST is shipped with WebSphere Application Server version 6.0 as free tool.

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

Related tasks

"Managing application profiles" on page 1355

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*.

Related tasks

"Managing application profiles" on page 1355

Related reference

"Application profile collection" on page 1359
Use this page to view application profiles and manage tasks associated with 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

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**.

Related tasks

"Managing application profiles" on page 1355

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*.

Related tasks

"Managing application profiles" on page 1355

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.

Related information

Java theory and practice: Thread pools and work queues

Asynchronous beans:

An asynchronous bean is a Java object or enterprise bean that can be executed 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. (Support for application profiling context is deprecated.)
- Work areas

Asynchronous bean interfaces

Three 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 Javadoc.

Timer listener

This interface is an object that implements the commonj\timers\TimerListener interface. Timer listeners are called when a high-speed transaction timer expires. For more information on the TimerListener interface, see the Javadoc.

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 Javadoc.

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 Javadoc.

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 Javadoc.

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 run-time environment 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 execute some work within this context. The `WorkWithExecutionContext` object can be executed 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 executed 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 .

Related concepts

Work objects
A work object is a type of asynchronous bean used by application components to run code in parallel or in a different J2EE context.

“Work managers” on page 1365
A work manager is a thread pool created for J2EE applications that use asynchronous beans.

References
References are logical names used to locate external resources for enterprise applications. References are defined in the application's deployment descriptor file. At deployment, the references are bound to the physical location (global JNDI name) of the resource in the target operational environment.

Related tasks

Developing asynchronous scopes

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 topic.

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 topic 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 JVM only if an application within that Java virtual machine (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.

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)
: 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 XD 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)
WorkCompleted Exception	(Extends WorkException)	WorkCompleted Exception	(Extends WorkException)
WorkRejected Exception	(Extends WorkException)	WorkRejected Exception	(Extends WorkException)

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

Work manager examples

Table 17. Look up work manager

Asynchronous beans	**CommonJ**

Table 17. Look up work manager (continued)

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 18. 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 19. Submit the work

Asynchronous beans	CommonJ

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 20. Create a timer manager

Asynchronous beans	CommonJ

Table 20. Create a timer manager (continued)

`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 21. Fire the timer

Asynchronous beans	CommonJ

`// 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);`

Related tasks

"Configuring work managers" on page 1375

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 22. Set up the timer listener

Asynchronous beans	CommonJ

Table 22. 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;
	}
	}

Related tasks

"Configuring timer managers" on page 1372

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...
 }
}
```

Related concepts

"Asynchronous beans" on page 1362
An asynchronous bean is a Java object or enterprise bean that can be executed asynchronously by a Java 2 Platform Enterprise Edition (J2EE) application, using the J2EE context of the asynchronous bean creator.

References
References are logical names used to locate external resources for enterprise applications. References are defined in the application's deployment descriptor file. At deployment, the references are bound to the physical location (global JNDI name) of the resource in the target operational environment.

Configuring timer managers

If you are not familiar with timer managers, review the conceptual section, Timer managers, in the Asynchronous beans topic.

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.

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. Click **New**.
4. Specify the following required properties:
   Name The display name for the timer manager.
   JNDI Name
   The 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.

A timer manager contains a pool of threads bound into JNDI.

To view this administrative console page, click **Resources** > **Asynchronous beans** < **Timer managers**.

Related tasks

“Configuring timer managers” on page 1372

Name:

The name by which the timer manager is known for administrative purposes.

Data type String

JNDI Name:

The JNDI name used to look up the timer manager in the name space.

Data type String

Description:

A description of this timer manager for administrative purposes.

Data type String

Category:

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.

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 configuration location for the configuration file.

Name:

The name by which the timer manager is known for administrative purposes.

Data type String

JNDI Name:

The JNDI name used to look up the timer manager in the namespace.

Data type String

Description:

A description of this timer manager for administrative purposes.

Data type String

Category:

A string that can be used to classify or group this timer manager.

Data type String

Default Transaction Class:

Specifies the transaction class that is used for WLM workload classification of non-daemon work that is not already associated with a service class.

Data type String
Range 0-8 characters

Service Names:

A list of service names on which this timer manager is made available.

Asynchronous beans can inherit J2EE context information by enabling one or more J2EE service contexts on the timer manager resource in the WebSphere administration 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 global 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:

The maximum number of threads that are used for timers.

Data type Integer

Configuring work managers

If you are not familiar with work managers, review the conceptual topic, 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. 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. Click **New**.
4. Specify the required properties for work manager settings.
   Name The display name for the work manager.
   JNDI Name
   The 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 order of the priority for threads available in the thread pool.

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 maximum number of scheduled work objects in this work request queue. The default value is 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.
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.

Related tasks

“Using asynchronous beans” on page 1361

Work manager collection:

Use this page to view the collection properties of work managers.

A work manager contains a pool of threads bound into the Java Naming and Directory Interface.

To view this administrative console page, click **Resources > Asynchronous beans > Work managers**.

Related tasks

“Configuring work managers” on page 1375

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

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.

A work manager contains a pool of threads bound into the 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 configuration 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 a value is not specified, then 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. The thread pool gets work from this queue. If you do not specify a value, 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 service names on which this work manager is made available.

Asynchronous beans can inherit J2EE context information by enabling one or more J2EE service contexts on the work manager resource in the WebSphere administration 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. This J2EE 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 is 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 global 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:

Number of alarm threads — Specifies the number of alarm threads available to the work manager for running work. The default value is 2.

Minimum number of threads — Specifies the minimum number of threads available in this work manager for running work.

Maximum number of threads — Specifies the maximum number of threads available in this work manager for running work.

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.

Default transaction class:

Specifies the transaction class name used to classify work run by this work manager instance when the z/OS Work Load Manager Service class information is not contained in the work context information.

Data type — String

Range — 0-8 characters

Daemon transaction class:

Specifies the transaction class name used to classify ″daemon″ work initiated by this work manager instance.

Data type — String

Range — 0-8 characters

Dynamic cache

Task overview: Using the dynamic cache service to improve performance

Use the dynamic cache to improve application performance by caching the output of servlets, commands, and JavaServer Pages (JSP) files.

On the z/OS platform, WebSphere Application Server, Version 4.0.1, supported the configuration of dynamic servlet caching through the use of a `servletcache.xml` file. For migration purposes, this file is still supported by this release. To utilize the new and improved functionality of the dynamic cache service in this release, you must configure your cache policy using the new `cachespec.xml` format. See “Configuring cacheable objects with the cachespec.xml file” on page 1400 for more information.

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 1384 for more information.
2. Configure the type of caching that you are using.
   - “Configuring servlet caching” on page 1386.
   - “Configuring Edge Side Include caching” on page 1392.
   - “Configuring command caching” on page 1413.
   - “Example: Caching Web services” on page 1380.

- "Configuring the Web services client cache" on page 1414.

3. You can monitor the results of your configuration using the dynamic cache monitor. For more information, see "Displaying cache information" on page 1427.
4. If you have any problems with your configuration, see Troubleshooting the dynamic cache service.

To use the DistributedMap and DistributedObjectCache interfaces for the dynamic cache, see "Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache" on page 1421.

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.

Example: Caching Web services: 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 Simple Object Access Protocol (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.

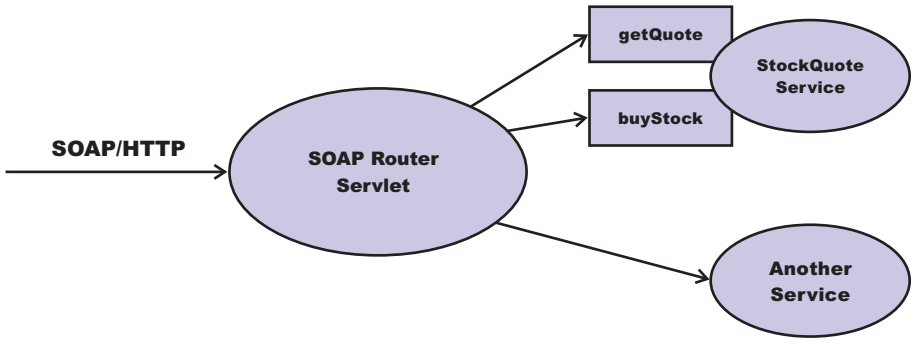
Message example 2 illustrates a SOAP message for a BuyQuote operation. While message 1 is cacheable, this message is not, because it updates the back end database.

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>
```

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>
```

Enabling the dynamic cache service

Enable the dynamic cache service to improve application performance by caching the output of servlets, Web services, and WebSphere 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 1400 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 1386 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 1389 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**.

Related concepts

"Dynamic cache" on page 1380
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.

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 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.

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, `$install_root/temp/node/servername/_dynacache/cacheJNDIname` is used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, `$install_root/diskoffload` generates the location as `$install_root/diskoffload/node/servername/cacheJNDIname`. This value is ignored if disk offload is not enabled.

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

Enable cache replication:

Use cache replication to have cache entries copied to multiple application servers that are configured in the same replication domain.

Full group replication domain:

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 application server.

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. On the other hand, 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.
- **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 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) means send the cache entries immediately. Setting this property to a value greater than `0` (zero) causes a ″batch″ push of all cache entries that are created or modified during the time period. The default is `0` (zero).

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 1384.

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″ in the informaiton center for more information.

Define the cache policy for your servlets by “Configuring cacheable objects with the cachespec.xml file” on page 1400.

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” on page 1386. To configure cache policies for your servlets, see “Configuring cacheable objects with the cachespec.xml file” on page 1400.

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.

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” for more information.

1. Enable servlet and JSP caching. Enabling servlet caching automatically enables Struts and Tiles caching. See “Configuring servlet caching” on page 1386 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 1400 for more information about cache policies.

See “Task overview: Using the dynamic cache service to improve performance” on page 1379 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.

On the z/OS platform, you can use disk offload if you have one servant in your application server. If there are multiple servants enabled in your configuration, do not enable disk offload. If you enable disk offload for an application server that has multiple servants, each servant attempts to save data to the same

location on the disk. For more information about enabling and disabling multiple servants, see ″Migrating V6.0 servers from multi-broker replication domains to data replication domains″ in the information center.

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 `htodCleanupFrequency` to configure the com.ibm.ws.cache.CacheConfig.htodCleanupFrequency 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.htodCleanupFrequency) 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.

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.

Tune the delay offload function

Use these properties to tune the delay offload function for the disk cache. 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.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.

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

com.ibm.ws.cache.CacheConfig.htodDelayOffload

Use this property to specify if extra memory buffers should be used in memory for dependency IDs and templates to delay disk offload and to minimize input and output operations to the disk. This property is enabled by default. However, consider disabling this property if your cache IDs are larger than 100 bytes because this option might use too much memory when it buffers your data. If you set this property to `false`, all the cache entries are copied to disk immediately after they are removed from the memory cache.

Default	true

Configuring Edge Side Include caching:

Edge Side Include (ESI) is configured through the `plugin-cfg.xml` file.

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.

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 z/OS, the esiInvalidationMonitor property must always be set to false. Therefore, this third option is not available on the z/OS platform.

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 ″“Configuring alternate URL.”″

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 cacheable fragments. However, for a fragment to be edge cacheable, you must be able to externally request it from the application server. 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 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 cacheable 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 1402.

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 Java Server Pages (JSP) `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 "install_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. 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.

b. 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.

c. 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 "*install_root*\IBMHttpServer\logs\afpalog" V-ECLF
      - LoadModule afpaplugin_module *install_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 *install_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 above 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 cache replication

Use this task to improve performance by configuring the data replication service (DRS) to replicate data from the dynamic cache service across the consumers in a replication domain.

See “Cache replication” on page 1400 for more information about replicating data for the dynamic cache service.

You should have a replication domain created for the dynamic cache service. Configure a different replication domain for each type of consumer of the replication domain. For example, configure two different replication domains for dynamic cache and session manager. There are two ways to configure replication domains:

- To create replication domains manually, click **Environment > Replication domains** in the administrative console.
- To create a new replication domain automatically when you create a cluster, click **Servers > Clusters > New** in the administrative console.

For more details about configuring replication domains, see ″Replicating data across application servers in a cluster″.

Do not use the default value of a single replica for the **Number of replicas** for dynamic cache replication domains. Instead, use a full group replica for any replication domains that you configure for dynamic cache.

Note: If you configured cache replication with a previous version of WebSphere Application Server, review ″Migrating V6.0 servers from multi-broker replication domains to data replication domains″ to learn about the new type of replication domains that you should use with the current version of WebSphere Application Server.

Use this task to improve performance in a clustered environment by enabling the data replication service (DRS) to replicate cached data across the servers in a cluster. Invalidations of cache entries are sent across the cluster to keep the cached data consistent and valid.

1. In the administrative console, click **Servers > Application servers >** *server_name* **> Container services > Dynamic cache service**.
2. To enable replication, select **Enable cache replication**.
3. Choose a replication domain. Use different replication domains for each type of consumer. For example, dynamic cache should use a different replication domain than session manager. The only replication domains that you can select in this panel include replication domains that are configured to use full-group replication. In a full-group configuration, every cache entry is replicated to every other cache that is configured in the servers that are in the replication domain. If none of the replication domains in your configuration meet these requirements, the list is empty. In this case, create a replication domain or alter an existing replication domain so that you have a replication domain that can perform full-group replication. See ″Replicating data across application servers in a cluster″ for more information.
4. Define the dynamic cache replication settings. Click **Enable cache replication**. On this page, you can define when and how often data is replicated across the dynamic cache replication domain. For more information about these settings, see “Dynamic cache service settings” on page 1384.
5. To enable cache replication on a base server that has multiple servants configured, you must define the following JVM custom properties.
   a. In the administrative console, click **Servers > Application servers >***server_name* **> Java and process management > Process definition > Servant > Java Virtual Machine > Custom properties > New**.

b. Add the new property name as `DynacacheEnableUnmanagedServerReplication` and set the value to `true`.

c. Add another JVM property to specify the global sharing policy for the base z/OS server. Click **New** and add a new JVM custom property that is called `DynacacheUnmanagedServerReplicationType`. Use one of the following values:

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.
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.
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.
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.

If you do not define one of these values, the default is NOT_SHARED.

6. Define sharing policies in the `cachespec.xml` file.

   To use cache replication among multiple servants on a base server, you must create a cachespec.xml file to define the cacheable objects that you want to create.You can override the global sharing policy by specifying a specific sharing policy in the cache policy. For example, if your global policy is to use Push only, you can change the sharing policy of a specific cache entry by making this change to your cache policy:

```
<cache-entry>
 <sharing-policy>not-shared</sharing-policy>
 <class>servlet</class>
 <name>/app</name>
 <cache-id>
  <component id="action" type="parameter">
   <value>portfolio</value>
   <required>true</required>
  </component>
  <component id="JSESSIONID" type="cookie">
   <required>true</required>
  </component>
  <property name="EdgeCacheable">true</property>
 </cache-id>
</cache-entry>
```

For more information about the sharing policies that can be defined in the cachespec.xml file, see "Cachespec.xml file" on page 1402.

Cache entries copy to the other application servers in the configured replication domain.

Use the cache monitor to view the contents of the cache. See "Displaying cache information" on page 1427 for more information.

Cache replication:

With replication, data is generated one time and copied or replicated to other servers in the cluster, saving time and resources. Caching in a cluster has additional concerns. In particular, the same data can be required and generated in multiple places. Also, the permission the resources need to generate the cached data can be restricted, preventing access to the data.

Cache replication deals with these concerns by generating the data one time and copying it to the other servers in the cluster. It also aids in cache consistency. Cache entries that are not needed are removed or replaced.

The data replication configuration can exist as part of the Web container dynamic cache configuration accessible through the administrative console, or on a per cache entry basis through the cachespec.xml file. With the cachespec.xml file, you can configure cache replication at the Web container level, but disable it for a specific cache entry.

Cache replication has some unique behavior on the z/OS platform. You can configure cache replication on a base server that has multiple servants enabled or on servers that are in a clustered environment. If you enable cache replication in a clustered environment, the replication occurs among all of the servants even if only a single server in the cluster is active.

Cache replication can take on three forms:
- **PUSH** - Send out new entries, both ID and data, and updates to those entries.
- **PULL** - Requests data from other servers in the cluster when that data is not locally present. This mode of replication is not recommended.
- **PUSH/PULL** - Sends out IDs for new entries, then, only requests from other servers in the cluster entries for IDs previously broadcast. The dynamic cache always sends out cache entry invalidations.

You can also perform a batch update. Specifically, for PUSH or PUSH/PULL, the dynamic cache broadcasts the update asynchronously, based on a timed interval rather than sending them immediately when they are created. Invalidations are sent immediately. Distribution of invalidations prevents stale data from residing in a cluster. For more information about configuring cache replication, see "Configuring cache replication" on page 1398 and "Dynamic cache service settings" on page 1384.

Configuring cacheable objects with the cachespec.xml file

Enable the dynamic cache. See "Enabling the dynamic cache service" on page 1384 for more information.

Use this task to define cacheable objects inside the cachespec.xml, found inside the Web module WEB-INF or enterprise bean META-INF directory.

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 *<install_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 1402 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 1402 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 1402 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" on page 1402 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 1400 for more information. You must have servlet caching enabled in the web container. See “Configuring servlet caching” on page 1386 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 96 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 1427 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 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 a part of a cache instance that is other than 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 then 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
- 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</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 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>
```

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 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 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.

Note: 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 created from this object. If this element is not present, a not-shared value is assumed. On the z/OS platform, you can enable replication between servants in a base application server by using the DynacacheEnableUnmanagedServerReplication and DynacacheUnmanagedServerReplicationType JVM custom properties. When enabling 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 “Configuring cache replication” on page 1398 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>
```

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. 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	Specifies the external cache name. The external cache name needs to match the external cache group name.

consume-subfragments	Servlet or Web service	True or false. 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. This 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.
do-not-consume	Servlet or Web service	True or false. 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 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. 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	True or false. 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.

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 defined in the cache policy of each cache entry. Each cache entry can have multiple cache ID rules that are executed in order until either:
- A rule returns a non-empty cache ID, or
- No more rules are left to execute.

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:

```
<cache-id>component*| timeout? | inactivity? | priority? | property* | idgenerator? | metadatagenerator?</cache-id>
```

component sub-element

Use the component sub-element to generate a portion of the cache ID. The component sub-element 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	Retrieves the named parameter value from the request object
parameter-list	servlet	Retrieves a list of values for the named parameter
session	servlet	Retrieves the named value from the HTTP Session
cookie	servlet	Retrieves the named cookie value
attribute	servlet	Retrieves the named request attribute
header	servlet and Web service	Retrieves the named request header
pathInfo	servlet	Retrieves the pathInfo from the request
servletpath	servlet	Retrieves the servlet path
locale	servlet	Retrieves the request locale
requestType	servlet	Retrieves the HTTP request method from the request.

Type	Valid classes	Meaning
tiles_attribute	servlet	Retrieves the value of an attribute from a tile.
SOAPEnvelope	Web service and Web services client cache	Retrieves the SOAPEnvelope from a Web services request. An ID attribute of `Hash` uses a Hash of the SOAPEnvelope, while `Literal` uses the SOAPEnvelope 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
operation	Web services client cache	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, the value should be formatted as namespaceOfOperation:nameOfOperation
part	Web services client cache	An input message part of 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>`

- Use the `ignore-value` attribute to specify whether or not to use the value returned by this component in cache ID formation. This is an optional attribute 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 that multiple dependency IDs (or invalidation IDs) should be generated 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 above 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 above cache policy generates the following component to be used 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. 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 getUser().getUserInfo().name
- Use the **required** element to specify whether or not this component must return a non-null value for this cache ID for it 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>`, but instead prescribes the defined values from caching. You can use multiple `<not-value>` elements when there is more than one invalid value. For example:

  ```
  <component id="getUser" type="method">
  <required>true</required>
  <not-value>blue</not-value>
  <not-value>red</not-value></component>
  ```

The component sub-element can have either a `method` and a `field` element, a `value` element, or a `not-value` element. The `method` and `field` elements apply only to commands. The following example illustrates the attributes of a component sub-element:

```
<component id="isValid" type="method" ignore-value="true"><component>
```

timeout sub-element

The timeout sub-element 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. If 0, or a negative value is specified, the cache entry is kept indefinitely.

inactivity sub-element

The inactivity sub-element is used to specify a time-to-live (TTL) value for the cache entry based on the last time the cache entry was accessed. It is a sub-element of <cache-id>.

```
<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 sub-element

Use the priority sub-element 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, it 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 sub-element

Use the property sub-element 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 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> classname </idgenerator>
```

Classname 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 defines properties like timeout, inactivity, external caching properties or dependencies. An example of the `metadatagenerator` element follows:

```
<metadatagenerator> classname </metadatagenerator>
```

In this example, classname 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 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 execute 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 defined in the cache policy of each cache entry. The following illustrates an invalidation in the cache policy:

```
<invalidation>component* | invalidationgenerator? </invalidation>
```

invalidationgenerator sub-element

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>classname</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. However, in this case, 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: To write a command interface, extend one or more of the three interfaces included in the command package. The base interface for all commands is the Command interface. This 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:

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 1379. 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 1400 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 1420.

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 366. 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: <install_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 1427 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 1414.

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 1384 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 1423.

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 Managing shared libraries and “Class loader settings” on page 31 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=${WAS_INSTALL_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.

To learn how to share cached objects in a clustered environment, see “Sharing cached objects in a clustered environment.”

Sharing cached objects in a clustered environment:

In a clustered environment, the content you place in cache might be shared with other servers in the cluster. The content might also be off-loaded to disk. If you intend to have the cached objects shared or off-loaded to disk, you must make these particular objects serializable. If the objects you place in cache are non-serializable, you must specify that the sharing policy for these objects is ″not shared″. The DistributedMap interface Reference: Generated API documentation contains information about how to specify the sharing policy for a cached object. Specifying a sharing policy other than ″not shared″ for non-serializable objects can result in poor system performance.

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 - 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

Disk offload location:

Specifies the directory that is used for disk offload.

If disk offload location is not specified, the default location, $*install_root*/temp/*node*/*servername*/_dynacache/*cacheJNDIname* is used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, $*install_root*/diskoffload generates the location as $*install_root*/diskoffload/*node*/*servername*/*cacheJNDIname*. This value is ignored if enableDiskOffload is false.

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

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) means send the cache entries immediately. Setting this property to a value greater than `0` (zero) causes a ″batch″ push of all cache entries that are created or modified during the time period. The default is `0` (zero).

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.

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.

On the z/OS platform, the cache monitor provides information on the cache in the servant to which your browser connects to interact with the monitor. In an environment with multiple servants, the cache monitor provides a partial view of caching activity.

1. Use the administrative console to install the cache monitor application from the *install_root*`/installableApps` directory. The application is named `CacheMonitor.ear`. For more information about installing applications, see “Installing application files with the console” on page 41. Install the cache monitor onto the application server you are trying to monitor.

   On the z/OS platform, the cache monitor application must be installed on the *default_host* (908*x*).
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. On z/OS, the Edge Statistics page is not supported.
   - View the Disk Offload page to view 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.

In the z/OS platform, Cache monitor provides information on the cache in the servant to which your browser connects to interact with the monitor. In an environment that has multiple servants, cache monitor provides a partial view of caching activity.

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 1402 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.
- **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, allowing new entries to enter the cache service. With disk offload the removed cache entries are copied disk for future access. You can view of the content that is copied to disk that corresponds to the view of 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.

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 "Displaying cache information" on page 1427 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 "Configuring servlet caching" on page 1386 for more information.
Disk Offload Enabled	Specifies if entries that are being removed from the cache are saved to disk. See "Configuring dynamic cache disk offload" on page 1389 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 “Dynamic cache service settings” on page 1384 to increase the cache size and “Configuring cacheable objects with the cachespec.xml file” on page 1400 for information about configuring cache policies.

By using the cache monitor application, you optimized the performance of the dynamic cache service.

See “Task overview: Using the dynamic cache service to improve performance” on page 1379 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 1384 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 533 for more information.

1. Enable servlet caching. See “Configuring servlet caching” on page 1386 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.

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

Disk offload location:

Specifies the directory used for disk offload.

If disk offload location is not specified, the default location, $*install_root*/temp/*node*/*servername*/_dynacache/*cacheJNDIname* will be used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, $*install_root*/diskoffload generates the location as $*install_root*/diskoffload/*node*/*servername*/*cacheJNDIname*. This value is ignored if disk offload is not enabled.

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

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.

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) means send the cache entries immediately. Setting this property to a value greater than `0` (zero) causes a ″batch″ push of all cache entries that are created or modified during the time period. The default is `0` (zero).

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 1384 and “Configuring servlet caching” on page 1386 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 1379 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.

For your WebSphere application, you can define an EJB query in 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.

Before using EJB query, familiarize yourself with query language concepts, starting with the topic, EJB Query Language.

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.

The SELECT clause is optional in order to maintain compatibility with WebSphere Application Server Version 4.

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 23. 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 24. 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)

Table 24. EmpBean schema (continued)

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 1454. 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 identified either by an abstract schema name and an identification variable, called a range variable, or by a collection member declaration that identifies a multivalued relationship and an identification variable.

Conceptually, the semantics of the query is to first form a temporary collection of tuples R. Tuples are composed of elements from the collections identified in the FROM clause. Each tuple contains one element from each of the collections in the FROM clause. All possible combinations are formed subject to the constraints imposed by the collection member declarations. If any schema name identifies a collection for which there are no records in the persistent store, then the temporary collection R will be empty.

Example: FROM clause

`DeptBean` contains records 10, 20 and 30 in the persistent store. `EmpBean` contains records 1, 2 and 3 that are related to department 10 and records 4, 5 that are related to department 20. Department 30 has no related employees.

```
FROM DeptBean d, EmpBean e
```

This forms a temporary collection R that contains 15 tuples.

```
FROM  DeptBean d,  DeptBean d1
```

This forms a temporary collection R that contains 9 tuples.

```
FROM  DeptBean d,  IN (d.emps) AS e
```

This forms a temporary collection R that contains 5 tuples. Department 30 because it contains no employees will not be in R. Department 10 will be contained in R three times and department 20 will be contained in R twice.

After forming the temporary collection the search conditions of the WHERE clause will be applied to R and this will yield a new temporary collection R1. The ORDER BY and SELECT clauses are applied to R1 to yield the final result set.

An identification variable is a variable declared in the FROM clause using the operator IN or the optional AS.

```
FROM DeptBean AS d,  IN (d.emps) AS e
```

is equivalent to:

```
FROM DeptBean d, IN (d.emps) e
```

An identification variable that is declared to be an abstract schema name is called a range variable. In the query above ″d″ is a range variable. An identification variable that is declared to be a multivalued path expression is called a collection member declaration. ″d″ and ″e″ in the example above are collection member declarations.

Note that the following path expression is illegal as a collection member declaration because it is not multivalued:

```
e.dept.mgr
```

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.

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

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
```

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.

A scalar-subselect is a subselect that returns a single value.

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
```

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
```

The following is a valid dynamic query:

```
SELECT OBJECT(e), e.salary+e.bonus as total_pay FROM  EmpBean e ORDER BY  2 DESC
```

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 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.

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 :=select_clause_dynamic from_clause [where_clause]
   [group_by_clause] [having_clause] [order_by_clause]

from_clause::=FROM identification_variable_declaration
              [,identification_variable_declaration]*

identification_variable_declaration::=collection_member_declaration |
     range_variable_declaration

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) |
        AVG([ALL|DISTINCT] expression) |
        BIGINT(expression) |
        CHAR({expression [,{ISO|USA|EUR|JIS}] )  |
        CONCAT (expression , expression ) |
        COUNT({[ALL|DISTINCT] 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] ) |
        MAX([ALL|DISTINCT] expression) |
        MICROSECOND( expression ) |
        MIN([ALL|DISTINCT] expression) |
        MINUTE ( expression ) |
        MOD (expression, expression) |
        MONTH( expression ) |
        REAL( expression) |
        SECOND( expression ) |
        SMALLINT( expression )  |
        SQRT (  expression) |
        SUBSTRING( expression, expression[, expression]) |
        SUM([ALL|DISTINCT] expression) |
```

```
TIME( expression ) |
TIMESTAMP( expression ) |
UCASE ( expression) |
YEAR( expression )
```

Comparison of EJB 2.1 specification and WebSphere query language: WebSphere Application Server Version 6.0 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 `<WAS_HOME>/installableApps` 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.

Related concepts

“EJB query language” on page 1436

“Path expressions” on page 1440

Related reference

“Example: EJB queries” on page 1436

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.

  Related concepts

  “EJB query language” on page 1436

  Related tasks

  “Using EJB query” on page 1435

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 174 and “Concurrency control” on page 175.

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.

Related concepts

"Access intent service" on page 179

"Dynamic query performance considerations" on page 1459

Related tasks

"Using the dynamic query service" on page 1456

Related information

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 runtime, rather than defining them at deployment time. By using dynamic query you gain the flexibility of queries defined at runtime 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: Internationalizing applications

An application that can present information to users according to regional cultural conventions is said to be *internationalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In an internationalized 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.

This product supports internationalization through use of its localizable-text API and internationalization service.

- 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 1466.
- 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 1466.

Internationalization:

An application that can present information to users according to regional cultural conventions is said to be *internationalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In an internationalized 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.

Historically, the creation of internationalized 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 internationalize a much wider variety of applications. This trend requires making internationalization 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 internationalized, 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.

Internationalization 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 1466.

Internationalization: Resources for learning: Use the following links to find relevant supplemental information about internationalization. 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.
- International Components for Unicode for Java

  The portal site for IBM’s open-source API to extend basic Unicode support in Java application components.

Programming specifications
- Java 2 SDK, Standard Edition Documentation: Internationalization

  The Java internationalization documentation from Sun Microsystems, including a list of supported locales and encodings.
- 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, Web Services Internationalization Task Force

The task force of the W3C's Internationalization 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 product supports the maintenance and deployment of centralized message catalogs for the output of properly formatted, language-specific (*localized*) interface strings.

This topic summarizes the steps involved in implementing message catalogs through the localizable-text API.

1. Identify localizable text in your application. See ″Identifying localizable text″ in the information center.
2. Create the message catalogs that are necessary for the locales to be supported by your application. See ″Creating message catalogs″ in the information center.
3. In your application code, compose the language-specific strings for output. See ″Composing language-specific strings″ in the information center.
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. See ″Preparing the localizable-text package for deployment″ in the information center. 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)

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.

This topic summarizes the steps involved in using the internationalization service.

1. If you have an application that uses the WebSphere Application Server Version 4.0 internationalization service, migrate your application as needed. See ″Migrating internationalized applications″ in the information center.
2. Use the internationalization context API within application components to obtain or manage internationalization context. See ″Using the internationalization context API″ in the information center. 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.

3. Assemble internationalized applications. See ″Assembling internationalized applications″ in the information center.

   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.
4. 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.
5. 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 internationalization 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. For an understanding of how these differences impact application development, read “Internationalization” on page 1463.

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*. See ″Internationalization context: Propagation and scope″ in the information center. 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.

Administering the internationalization service

To use internationalization context in an Enterprise JavaBeans (EJB) application, the internationalization service must be enabled in the run-time 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, disable the service on all Java 2 Platform, Enterprise Edition (J2EE) clients. (By default, the service is disabled for server-side components.) Disabling the service eliminates 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. The service is disabled by default within WebSphere application servers. You enable the service by using either the administrative console or the wsadmin tool.
- Enable or disable the internationalization service for EJB clients. The service is enabled by default within the WebSphere Application Server client container.

Related concepts

“Internationalization service” on page 1467

Enabling tracing and logging

Enabling the internationalization service for servlets and enterprise beans:

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. See ″Starting and logging off the administrative console″ in the information center.
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. See ″Stopping servers″ and ″Starting servers″ in the information center.

Enabling the internationalization service for EJB clients:

By default, the internationalization service is enabled for use within Enterprise JavaBeans (EJB) client applications whenever the i18nctx.jar file is in the CLASSPATH setting that is constructed by the launchClient tool. The internationalization service is also enabled for Web service-enabled clients.

When invoking a Java client application, the launchClient tool sets the CLASSPATH to include the i18nctx.jar file and then activates the client container, which initializes, starts, and enables the service before delegating to the specified application.

To disable the service for all application server instances in your installation, remove thei18nctx.jar file from the *install_root*/lib directory. This action prevents the file from inadvertently being included in the CLASSPATH setting constructed by the launchClient tool.

To selectively disable the service, include the argument `-CCDI18NService.enable=false` or `-CCDI18NService.enable=no` when invoking the launchClient tool.

Related concepts

“Internationalization service” on page 1467

Related tasks

“Administering the internationalization service” on page 1468

“Enabling the internationalization service for servlets and enterprise beans” on page 1468

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, or to use this 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**.

Related concepts

“Internationalization service” on page 1467

Related tasks

“Administering the internationalization service” on page 1468

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.

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: The internationalization service issues one exception: java.lang.IllegalStateException. This 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 ”
- “The service is not started” on page 1471
- “Invalid context element” on page 1472
- “Missing context element” on page 1472
- “Invalid policy” on page 1472
- “Missing policy” on page 1472

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 Enabling tracing and logging.

The trace strings for internationalization 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 run-time 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 run-time problems:

No ORB support
: The service cannot obtain an instance of the object request broker (ORB). This condition is a fatal error. Examine the logs 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 logs 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. Review the logs for error conditions related to the activity service.

No AsynchBeans support
: The service cannot register into the asynchronous beans environment. This warning indicates that the asynchronous beans environment cannot support internationalization context. If the application server is supposed to support asynchronous beans, verify that the asynchbeans.jar and asynchbeansimpl.jar files exist in the class path, and review the trace log for any error conditions related to asynchronous beans.

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 Meta-data support
: The service cannot register with the meta-data service. This warning indicates that the internationalization service cannot process the internationalization policies within application deployment descriptors. Without meta-data 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 meta-data 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. For more information see “Administering the internationalization service” on page 1468.

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 Assembling internationalized applications.

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 meta-data support is unavailable, the service logs a message and applies the policy `[CMI, RunAsCaller]` on every servlet service method and EJB business method invocation.

For more information, see the following topics:
- Assembling internationalized applications
- ″Container internationalization attributes″
- ″Internationalization type″
- Migrating internationalized applications

Related concepts

Enabling tracing and logging

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. These 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 Pools**.
3. Define the name of the object pool manager. This name can be up to 30 ASCII characters long.
4. Assign the object pool manager a Java Naming and Directory Interface (JNDI) name.
5. Provide a description of this object pool manager.
6. Categorize 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.

Related tasks

"Using object pools" on page 1473

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 Messaging 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.

Related tasks

"Using object pools" on page 1473

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 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.

Related tasks

"Using object pools" on page 1473

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 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 >** *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.

Related tasks

"Using object pools" on page 1473

Pool class name:

The fully qualified class name of the objects that are stored in the object pool.

Data type String

Pool implementation class name:

The fully qualified class name of the CustomObjectPool implementation class for this 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.

Related tasks

"Using object pools" on page 1473

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**.

Related tasks

"Using object pools" on page 1473

Enable service at server startup:

Specifies whether the server attempts to start the object pool service.

Default
Selected

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: 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

Related tasks

"Using object pools" on page 1473

Scheduler

Using schedulers

Schedulers enable J2EE application tasks to run at a requested time. 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

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. 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 JavaBeans 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. The ″Developing and scheduling tasks″ 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. The ″Interoperating with schedulers″ 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`.

Related tasks

"Configuring schedulers" on page 1482

Developing and scheduling tasks

Interoperating with schedulers

Related reference

"Schedulers collection" on page 1484
Use this page to manage scheduler configurations.

Example: Stopping and starting scheduler daemons using Java Management Extensions API:

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."
```

Related concepts

"Scheduler daemon" on page 1480
A scheduler daemon is a background thread that searches for tasks to run in the database.

Related reference

Javadoc
This reference information is the generated API documentation describing various WebSphere Application Server application programming interfaces (APIs).

Related information

"Example: Dynamically changing scheduler daemon poll intervals using Java Management Extensions API"

Example: Dynamically changing scheduler daemon poll intervals using Java Management Extensions API:

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."
```

Related concepts

"Scheduler daemon" on page 1480
A scheduler daemon is a background thread that searches for tasks to run in the database.

Related reference

Javadoc
This reference information is the generated API documentation describing various WebSphere Application Server application programming interfaces (APIs).

Related information

"Example: Stopping and starting scheduler daemons using Java Management Extensions API" on page 1481

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.

Related tasks

Developing and scheduling tasks

Related information

WebSphere Enterprise Scheduler planning and administration guide

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.

Related tasks

Developing and scheduling tasks

Interoperating with schedulers

Related information

WebSphere Enterprise Scheduler planning and administration guide

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, 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 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.

Related tasks

"Creating a scheduler resource reference" on page 1490

"Configuring schedulers" on page 1482

Related reference

"Schedulers settings" on page 1485
Use this page to modify scheduler settings.

Schedulers collection:

Use this page to manage scheduler configurations.

To view this administrative console page, click **Resources > Schedulers**.

Related concepts

"Scheduler daemon" on page 1480
A scheduler daemon is a background thread that searches for tasks to run in the database.

Related tasks

"Configuring schedulers" on page 1482

"Creating a scheduler resource reference" on page 1490

Name:

The name by which this scheduler is known for administrative purposes.

Data type String

JNDI name:

The JNDI name for the scheduler.

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

Data source JNDI name:

Data source where persistent tasks will be 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

Table prefix:

The prefix to apply to all of the scheduler tables and indices. This can optionally include a schema name if the database requires one.

Multiple independent schedulers can share the same database if each instance specifies a different prefix string.

Data type String

Poll interval:

The interval at which the scheduler daemon polls the database.

Each poll operation can be expensive. 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 managers:

Specifies work managers used by this scheduler.

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 will use the ″Number Of Alarm Threads″ specified in the work manager, which affects the number 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. Configuring a scheduler with a specific work manager enables you to control how many tasks are actively running at a given time.

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 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. 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 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*.

Related concepts

"Scheduler daemon" on page 1480
A scheduler daemon is a background thread that searches for tasks to run in the database.

Related tasks

"Creating a scheduler resource reference" on page 1490

"Configuring schedulers" on page 1482

Name:

The name by which this scheduler is known for administrative purposes.

Data type String

JNDI name:

The JNDI name for the scheduler.

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:

A description of this scheduler for administrative purposes.

Data type String

Category:

A string that can be used to classify or group this scheduler.

Data type String

Data source JNDI name:

Data source where persistent tasks will be 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:

Alias to a user name and password used to access the data source.

Data type String

Table prefix:

The prefix to apply to all of the scheduler tables and indices. This can optionally include a schema name if the database requires one.

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:

The interval at which the scheduler daemon polls the database.

Each poll operation can be expensive. 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 managers:

Specifies work managers used by this scheduler.

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 affect the number 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: Although work managers and scheduler instances are configured at different scopes, schedulers 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 whether to use the define scheduler roles.

Schedulers require several user roles to plan for, develop, administer and operate the scheduler service: administrator, developer and operator. If checked, and global security is enabled, then administration roles are enforced when using scheduler JMX or APIs to create and modify tasks. If this option is not enabled, then all users can create and modify tasks.

Data type	check box
Default	unchecked

Range

- **Operator**--Calls any of the scheduler MBean or API methods and run any of the scheduler administrative console functions.
- **Monitor**--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. In order 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 desired.
   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 the scheduler daemons topic.

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:

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"

# Find our WorkManagerInfo object.
# If we don't have a DefaultWorkManager at the server scope,
# copy the one from the Node 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
}

set wrkMgrInfo [$AdminConfig getid /Cell:$cellName/Node:$nodeName/Server:$serverName/
WorkManagerProvider:WorkManagerProvider/WorkManagerInfo:DefaultWorkManager/]
if {$wrkMgrInfo == ""} {
    puts "Unable to find the DefaultWorkManager for server: $serverName.  Creating one."
    set wrkMgrInfo [$AdminConfig getid /Cell:$cellName/Node:$nodeName/WorkManagerProvider:WorkManagerProvider/
WorkManagerInfo:DefaultWorkManager/]
    if {$wrkMgrInfo == ""} {
       puts "Unable to find the DefaultWorkManager for node: $nodeName.  Aborting."
       exit
    }
    # Setup our DefaultWorkManager attributes
    set createAttrs [subst { \
        {category "[$AdminConfig showAttribute $wrkMgrInfo category]"} \
        {description "[$AdminConfig showAttribute $wrkMgrInfo description]"} \
        {isGrowable [$AdminConfig showAttribute $wrkMgrInfo isGrowable]} \
        {jndiName [$AdminConfig showAttribute $wrkMgrInfo jndiName]} \
        {maxThreads [$AdminConfig showAttribute $wrkMgrInfo maxThreads]} \
        {minThreads [$AdminConfig showAttribute $wrkMgrInfo minThreads]} \
        {name "[$AdminConfig showAttribute $wrkMgrInfo name]"} \
        {numAlarmThreads [$AdminConfig showAttribute $wrkMgrInfo numAlarmThreads]} \
        {serviceNames "[$AdminConfig showAttribute $wrkMgrInfo serviceNames]"} \
        {threadPriority [$AdminConfig showAttribute $wrkMgrInfo threadPriority]} }]
    set wrkMgrInfo [$AdminConfig create WorkManagerInfo $wrkMgrProv $createAttrs]

    puts "Created a DefaultWorkManager"

}
puts "Found a WorkManagerInfo"

# 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} \
  {workManagerInfo $wrkMgrInfo}}]

# Create the Scheduler
$AdminConfig create SchedulerConfiguration $schedProv $createAttrs
puts "Scheduler created"
```

```
# Save the configuration
$AdminConfig save

# Reload the configuration
set dshelper [$AdminControl queryNames type=DataSourceCfgHelper,process=$serverName,*]
$AdminControl invoke $dshelper reload
```

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 Web 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

Related tasks

Assembling applications
Application assembly consists of creating Java 2 Platform, Enterprise Edition (J2EE) modules that can be deployed onto application servers. The modules are created from code artifacts such as Web application archives (WAR files), resource adapter archives (RAR files), enterprise bean (EJB) JAR files, and application client archives (JAR files). This packaging and configuring of code artifacts into enterprise application modules (EAR files) or standalone Web modules is necessary for deploying the modules onto an application server.

Related reference

"Schedulers collection" on page 1484
Use this page to manage scheduler configurations.

Creating the database for schedulers:

Each scheduler requires a database in which to store its persistent information. The choice of database and location should be determined by the application developer and server administrator.

Schedulers use this database for storing tasks and then running them. 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 ″Java Management Extensions (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:

Your database system must be installed and available.

It is important to know that the scheduler uses this database for storing and running tasks. 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 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 database is created and ready for you to create scheduler tables.

Related tasks

"Creating scheduler tables using DDL files" on page 1502

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.

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 Cloudview utility supplied with the Cloudscape system to create the database. For example, use the database name, `scheddb`. The embedded version of Cloudscape supports only one local connection. If 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.
5. Exit the Cloudview utility.

The Cloudscape database for the scheduler service exists.

Related tasks

"Configuring Cloudscape Version 5.1.60x" on page 637

Related reference

"Vendor-specific data sources minimum required settings" on page 561

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.

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.

Related reference

"DB2 tuning parameters" on page 638
"Vendor-specific data sources minimum required settings" on page 561

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.

1. You must have already installed 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.

Related reference

"Vendor-specific data sources minimum required settings" on page 561

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.

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.

Related reference

"Vendor-specific data sources minimum required settings" on page 561

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.

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.

Related reference

"Vendor-specific data sources minimum required settings" on page 561

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.

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.

Related reference

"Vendor-specific data sources minimum required settings" on page 561

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.

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.

Related reference

"Vendor-specific data sources minimum required settings" on page 561

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 deployment manager or 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. For example, if a scheduler is configured at the server scope, the data source at the node scope, and the verify tables operation is being run from a deployment manager, the deployment manager attempts to connect to server1 and run the operation there. If the server is unavailable, then it attempts to verify at the node agent level, and then at the deployment manager level.

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.

Related tasks

“Creating scheduler tables using the administrative console” on page 1497

“Creating scheduler tables using scripting and Java Management Extensions” on page 1498

Related reference

“Schedulers settings” on page 1485
Use this page to modify scheduler settings.

Javadoc
This reference information is the generated API documentation describing various WebSphere Application Server application programming interfaces (APIs).

Scheduler table definition:

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	The value, **1**, if the task is suspended.
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.

Related tasks

"Creating scheduler databases" on page 1491

Creating 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.

Related tasks

Using the administrative console

“Creating and configuring a data source using the administrative console” on page 589

“Testing a connection with the administrative console” on page 616

“Configuring schedulers” on page 1482

Related reference

“Scheduler table management functions” on page 1494

“Scheduler table definition” on page 1495

Creating scheduler tables using scripting and Java Management Extensions:

The scheduler 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.

Related tasks

Using the administrative console

“Creating and configuring a JDBC provider and data source using the Java Management Extensions API” on page 602

“Testing a connection using wsadmin” on page 616

“Configuring schedulers” on page 1482

Related reference

Javadoc
This reference information is the generated API documentation describing various WebSphere Application Server application programming interfaces (APIs).

“Scheduler table definition” on page 1495

Related information

“Example: Using scripting to create and configure schedulers” on page 1488

Example: Using scripting to verify scheduler tables:

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 1482

“Configuring schedulers using Java Management Extensions” on page 1488

“Creating scheduler tables using scripting and Java Management Extensions” on page 1498

Related reference

“Scheduler table management functions” on page 1494

Example: Using scripting to 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 1482

“Configuring schedulers using Java Management Extensions” on page 1488

“Creating scheduler tables using scripting and Java Management Extensions” on page 1498

Related reference

“Scheduler table management functions” on page 1494

Example: Using scripting to drop 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 1482

“Configuring schedulers using Java Management Extensions” on page 1488

“Creating scheduler tables using scripting and Java Management Extensions” on page 1498

Related reference

“Scheduler table management functions” on page 1494

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.

Related tasks

“Creating scheduler databases” on page 1491

Related reference

“Scheduler table definition” on page 1495

Creating Cloudscape tables for schedulers:

This task requires you to configure a database and make it available. See the ″Creating Cloudscape databases for schedulers″ topic, for more information.

This topic describes how to create tables for schedulers on Cloudscape databases, using data definition language (DDL) or structured query language (SQL) files.

1. Open a command-line window.
2. Create the schema.
   a. Using a text editor, edit the script, *%install_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):

```
%install_root%\cloudscape\bin\embedded\ij.bat %install_root%\Scheduler\createSchemaCloudscape.ddl
```

On UNIX systems (using the example name, scheddb):

```
%install_root%/cloudscape/bin/embedded/ij.sh %install_root%/Scheduler/createSchemaCloudscape.ddl
```

The Cloudscape tables and schema for the scheduler exist.

Related tasks

"Creating Cloudscape databases for schedulers" on page 1492

"Configuring Cloudscape Version 5.1.60x" on page 637

Related reference

"Vendor-specific data sources minimum required settings" on page 561

Creating DB2 tables for schedulers:

This task requires you to configure a database and make it available. See the ″Creating DB2 databases for schedulers″ topic for more information.

This topic describes how to create tables for scheduler on DB2 databases, using data definition language (DDL) or structured query language (SQL) files.

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.

Related tasks

"Creating DB2 databases for schedulers" on page 1492

Related reference

"DB2 tuning parameters" on page 638

"Vendor-specific data sources minimum required settings" on page 561

Creating DB2 for z/OS tables for schedulers:

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

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.

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.

Related tasks

“Creating DB2 for z/OS databases for schedulers” on page 1493

Related reference

“Vendor-specific data sources minimum required settings” on page 561

Creating Informix tables for schedulers:

This task requires that you configure a database and make it available. See the ″Creating Informix databases for schedulers″ topic for more information.

This topic describes how to create tables for schedulers on Informix databases, using data definition language (DDL) or structured query language (SQL) files.

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.

Related tasks

“Creating Informix databases for schedulers” on page 1493

Related reference

"Vendor-specific data sources minimum required settings" on page 561

Creating Microsoft SQL Server tables for schedulers:

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.

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.

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 %install_root%\Scheduler
   ```

   On UNIX systems, enter:

   ```
   cd $install_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.

Related tasks

"Creating Microsoft SQL Server databases for schedulers" on page 1493

Related reference

"Vendor-specific data sources minimum required settings" on page 561

Creating Oracle tables for schedulers:

This task requires you to configure a database and make it available. See the ″Creating Oracle databases for schedulers″ topic for more information.

This topic describes how to create tables for schedulers on Oracle databases, using data definition language (DDL) or structured query language (SQL) files.

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.

      `%install_root%\Scheduler\createTablespaceOracle.ddl and %install_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.

Related tasks

"Creating Oracle databases for schedulers" on page 1494

Related reference

"Vendor-specific data sources minimum required settings" on page 561

Creating Sybase tables for schedulers:

This task requires you to configure a database and make it available. See the ″Creating Sybase databases for schedulers″ topic for more information.

This topic describes how to create tables for schedulers on Sybase databases, using data definition language (DDL) or structured query language (SQL) files.

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.

      ```
      <install_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.

Related tasks

"Creating Sybase databases for schedulers" on page 1494

Related reference

"Vendor-specific data sources minimum required settings" on page 561

Startup beans

Using startup beans

A startup bean is a session bean that is loaded when an application starts. 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. Use the home interface, com.ibm.websphere.startupservice.AppStartUpHome, to designate a bean as a startup bean.
2. Use the remote interface, com.ibm.websphere.startupservice.AppStartUp, to define start() and stop() methods on the bean.

   The startup bean start() method is called when the application starts and contains business logic to be run at 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() method is called when the application stops and contains business logic to be run at 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. Beans are stopped in the opposite order to their start priority.

View the startup beans service settings.

Related tasks

“Using asynchronous beans” on page 1361

Startup beans service settings:

Use this page to enable or disable startup beans on all applications within an Application Server. A startup bean is a special session bean with start() and stop() methods containing business logic that is run at module or application start and stop time. 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**.

Related tasks

"Using startup beans" on page 1508

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

Use the following steps to enable startup beans in the administrative console. This action enables Java 2 Platform Enterprise Edition (J2EE) applications to run business logic automatically, whenever an application starts or stops normally.

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.

Related tasks

"Using asynchronous beans" on page 1361

Related reference

"Startup beans service settings" on page 1508
Use this page to enable or disable startup beans on all applications within an Application Server. A startup bean is a special session bean with start() and stop() methods containing business logic that is run at module or application start and stop time. Startup beans are especially useful when used with asynchronous beans.

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.

Work area service - Overview: 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.

Related tasks

"Developing applications that use work areas" on page 1516

"Managing the work area service" on page 1525

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.

Related tasks

"Setting properties in a work area" on page 1519

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.

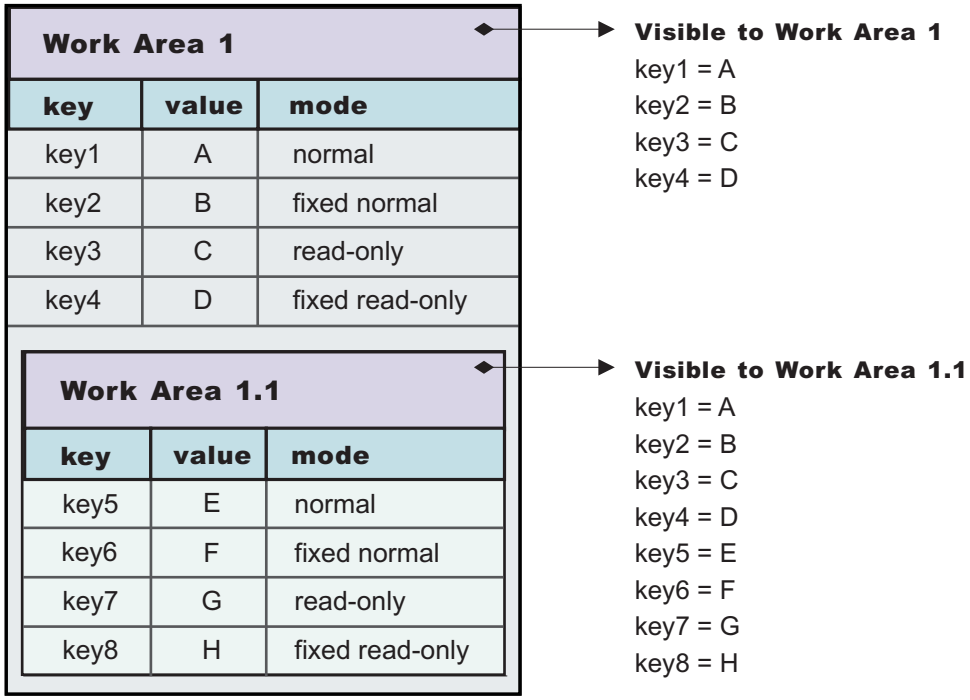


Figure 18. Defining new properties in nested work areas

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.

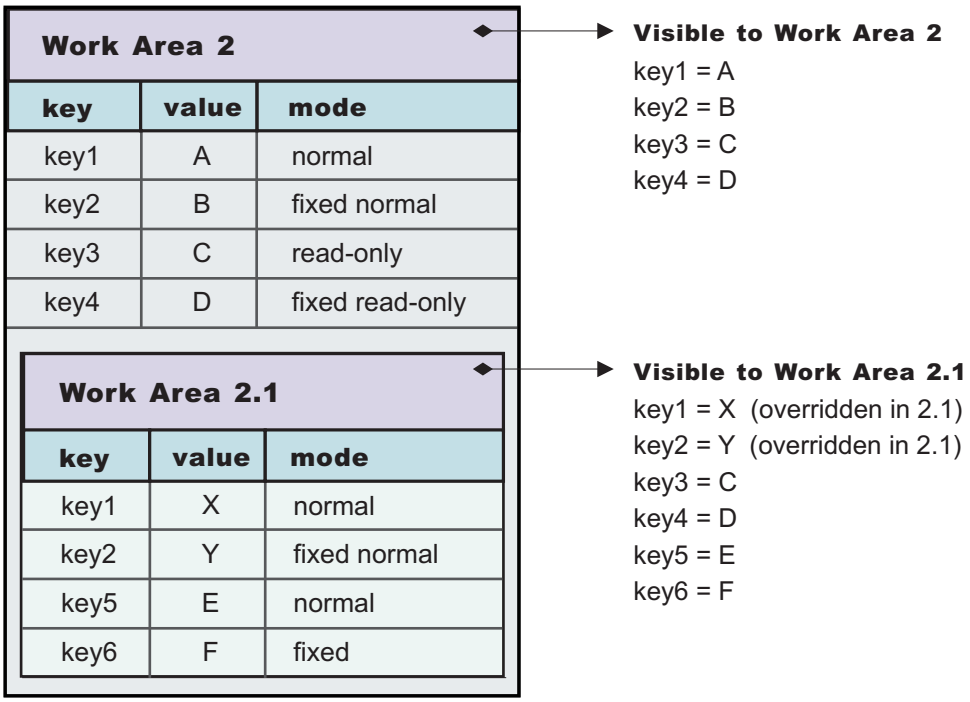


Figure 19. Redefining existing properties in nested work areas

Related concepts

"Work area property modes" on page 1510

Related tasks

"Setting properties in a work area" on page 1519

"Overriding work area properties" on page 1521

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.

Related concepts

"Work area property modes" on page 1510

"Work area partition service" on page 1528
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.

Related tasks

"Setting properties in a work area" on page 1519

"Overriding work area properties" on page 1521

Related reference

"Bidirectional propagation" on page 1534

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. 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.

Related concepts

"Work area property modes" on page 1510

Related tasks

"Setting properties in a work area" on page 1519

"Overriding work area properties" on page 1521

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 1528 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.

Developing applications that use work areas

Applications interact with the work area service by using the UserWorkArea interface and its implementation. This interface defines all of the methods used to create, manipulate, and complete work areas:

1. Access the partition by either:
   - "Accessing the work area service" on page 1518, to access the UserWorkArea partition.
   - "Accessing a user defined work area partition" on page 1536, to access a user defined work area.

   The following steps use the UserWorkArea partition as an example, however a user defined partition can be used in the same way.
2. Beginning a work area.
3. Setting properties in a work area.
4. Using a work area to manage local work.
5. Completing a work area.

An example application, the Work area SimpleSample application, is used throughout this documentation to illustrate these tasks

Related concepts

"Work area service - Overview" on page 1510

"Work area partition service" on page 1528
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.

Related tasks

"Managing the work area service" on page 1525

Related reference

"Accessing a user defined work area partition" on page 1536

"Example: Work area partition manager" on page 1531

UserWorkArea interface: Applications interact with the work area service by implementing the UserWorkArea interface. This interface, shown below, defines all of the methods used to create, manipulate, and terminate work areas:

```
package com.ibm.websphere.workarea;

public interface UserWorkArea {
   void begin(String name);
   void complete() throws NoWorkArea, NotOriginator;

   String getName();
   String[] retrieveAllKeys();
   void set(String key, java.io.Serializable value)
      throws NoWorkArea, NotOriginator, PropertyReadOnly;
   void set(String key, java.io.Serializable value, PropertyModeType mode)
      throws NoWorkArea, NotOriginator, PropertyReadOnly;
   java.io.Serializable get(String key);
   PropertyModeType getMode(String key);
   void remove(String key)
      throws NoWorkArea, NotOriginator, PropertyFixed;
}
```

Note: Enterprise JavaBeans (EJB) applications can use the UserWorkArea interface only within the implementation of methods in the remote interface; 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.

Exceptions

The work area service defines the following exceptions for use with the UserWorkArea interface:

NoWorkArea
Raised when a request requires an associated work area but none is present.

NotOriginator
Raised when a request attempts to manipulate the contents of an imported work area.

PropertyReadOnly
Raised when a request attempts to modify a read-only or fixed read-only property.

PropertyFixed
Raised by the remove method when the designated property has one of the fixed modes.

Example: WorkArea SimpleSample application: In this example, the client creates a work area and inserts two properties into the work area: a site identifier and a priority. The site-identifier is set as a read-only property; the client does not allow recipients of the work area to override the site identifier. This property consists of the key company and a static instance of a SimpleSampleCompany object. The priority property consists of the key priority and a static instance of a SimpleSamplePriority object. The object types are defined as shown in the following code example

```
public static final class SimpleSampleCompany {
   public static final SimpleSampleCompany Main;
   public static final SimpleSampleCompany NewYork_Sales;
   public static final SimpleSampleCompany NewYork_Development;
   public static final SimpleSampleCompany London_Sales;
   public static final SimpleSampleCompany London_Development;
}

public static final class SimpleSamplePriority {
   public static final SimpleSamplePriority Platinum;
   public static final SimpleSamplePriority Gold;
   public static final SimpleSamplePriority Silver;
   public static final SimpleSamplePriority Bronze;
   public static final SimpleSamplePriority Tin;
}
```

The client then makes an invocation on a remote object. The work area is automatically propagated; none of the methods on the remote object take a work area argument. On the remote side, the request is first handled by the SimpleSampleBean; the bean first reads the site identifier and priority properties from the

work area. The bean then intentionally attempts, and fails, both to write directly into the imported work area and to override the read-only site-identifier property.

The SimpleSampleBean successfully begins a nested work area, in which it overrides the client's priority, then calls another bean, the SimpleSampleBackendBean. The SimpleSampleBackendBean reads the properties from the work area, which contains the site identifier set in the client and priority set in the SimpleSampleBean. Finally, the SimpleSampleBean completes its nested work area, writes out a message based on the site-identifier property, and returns.

The implementation of this application is discussed in the topic, Developing applications that use work areas.

Accessing the work area service:

The work area service provides a JNDI binding to an implementation of the UserWorkArea interface under the name java:comp/websphere/UserWorkArea. Applications that need to access the service can perform a lookup on that JNDI name, as shown in the following code example:

```
import com.ibm.websphere.workarea.*;
import javax.naming.*;

public class SimpleSampleServlet {
  ...

  InitialContext jndi = null;
  UserWorkArea userWorkArea = null;
  try {
     jndi = new InitialContext();
     userWorkArea = (UserWorkArea)jndi.lookup(
        "java:comp/websphere/UserWorkArea");
  }
  catch (NamingException 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.

Related concepts

"Work area service - Overview" on page 1510

Related tasks

"Managing the work area service" on page 1525

Beginning a new work area:

Be sure that your client has a reference to the UserWorkArea interface, as described in the topic Accessing the work area service or a reference to a user defined partition as defined in "Accessing a user defined work area partition" on page 1536. The following steps use the UserWorkArea partition as an illustration. However a user defined partition can be used in the exact same way.

Use the begin method to create a new work area and associate it with the calling thread. A work area is scoped to the thread that began the work area and is not accessible by multiple threads. The begin method takes a string as an argument; the string is used to name the work area. The argument must not be null, which causes the java.lang.NullPointer exception to be raised. In the following code example, the application begins a new work area with the name SimpleSampleServlet:

```
public class SimpleSampleServlet {
...
   try {
      ...
      userWorkArea = (UserWorkArea)jndi.lookup(
         "java:comp/websphere/UserWorkArea");
```

```
   }
   ...

   userWorkArea.begin("SimpleSampleServlet");
   ...
}
```

The begin method is also used to create nested work areas; if a work area is associated with a thread when the begin method is called, the method creates a new work area nested within the existing work area.

The work area service makes no use of the names associated with work areas; You can name work areas in any way that you choose. Names are not required to be unique, but the usefulness of the names for debugging is enhanced if the names are distinct and meaningful within the application. Applications can use the getName method to return the name associated with a work area by the begin method.

Using a work area

Related concepts

“Nested work areas” on page 1511

Related tasks

“Completing a work area” on page 1524

“Retrieving the name of the active work area” on page 1521

Setting properties in a work area:

An application with a current work area can insert properties into the work area and retrieve the properties from the work area. The UserWorkArea interface provides two set methods for setting properties and a get method for retrieving properties. The two-argument set method inserts the property with the property mode of normal. The three-argument set method takes a property mode as the third argument. (See ″Setting property modes″, later in this topic.)

Both set methods take the key and the value as arguments. The key is a String; the value is an object of the type java.io.Serializable. None of the arguments can be null, which causes the java.lang.NullPointer exception to be raised.

The “Example: WorkArea SimpleSample application” on page 1517 uses objects of two classes, the SimpleSampleCompany class and the SimpleSampleProperty class, as values for properties. The SimpleSampleCompany class is used for the site identifier, and the SimpleSamplePriority class is used for the priority. These classes are shown in following code example:

```
public class SimpleSampleServlet {
   ...
   userWorkArea.begin("SimpleSampleServlet");

   try {
      // Set the site-identifier (default is Main).
      userWorkArea.set("company",
         SimpleSampleCompany.Main, PropertyModeType.read_only);

      // Set the priority.
      userWorkArea.set("priority", SimpleSamplePriority.Silver);
   }

   catch (PropertyReadOnly e) {
     // The company was previously set with the read-only or
     // fixed read-only mode.
     ...
   }

    catch (NotOriginator e) {
```

```
      // The work area originated in another process,
      // so it can't be modified here.
      ...
    }

    catch (NoWorkArea e) {
       // There is no work area begun on this thread.
       ...
    }

    // Do application work.
    ...
}
```

The get method takes the key as an argument and returns a Java Serializable object as the value associated with the key. For example, to retrieve the value of the company key from the work area, the code example above uses the get method on the work area to retrieve the value.

Setting property modes. The two-argument set method on the UserWorkArea interface takes a key and a value as arguments and inserts the property with the default property mode of normal. To set a property with a different mode, applications must use the three-argument set method, which takes a property mode as the third argument. The values used to request the property modes are as follows:

- **Normal**: PropertyModeType.normal
- **Fixed normal**: PropertyModeType.fixed_normal
- **Read-only**: PropertyModeType.read_only
- **Fixed read-only**: PropertyModeType.fixed_readonly

Related information

"Work area property modes" on page 1510

Using a work area to manage local work:

Be sure that your client has a reference to the UserWorkArea interface, as described in the topic "Accessing the work area service" on page 1518 or a reference to a user defined partition as defined in "Accessing a user defined work area partition" on page 1536. The following steps use the UserWorkArea partition as an illustration. However a user defined partition can be used in the exact same way.

In a business application that uses work areas, server objects typically retrieve the work area properties and use them to guide local work.

1. Retrieving the name of the active work area This step determines whether the calling thread is associated with a work area.
2. Overriding work area properties. Server objects can override client work area properties by creating their own, nested work area.
3. Retrieving properties from a work area
4. Retrieving a list of all keys in a work area
5. Querying the mode of a work area property
6. Deleting a work area property
7. Completing a work area

The server side of the "Example: WorkArea SimpleSample application" on page 1517 accepts remote invocations from clients. With each remote call, the server also gets a work area from the client if the client has created one. The work area is propagated transparently. None of the remote methods includes the work area on its argument list.

In the example application, the server objects use the work area interface for demonstration purposes only. For example, the SimpleSampleBean intentionally attempts to write directly to an imported work area, which creates the NotOriginator exception. Likewise, the bean intentionally attempts to mask the read only

SimpleSampleCompany, which triggers the PropertyReadOnly exception. The SimpleSampleBean also nests a work area and successfully overrides the priority property before invoking the SimpleSampleBackendBean. A true business application would extract the work area properties and use them to guide the local work. The SimpleSampleBean mimics this by writing a message that function is denied when a request emanates from a sales environment.

Related concepts

"Nested work areas" on page 1511

Retrieving the name of the active work area:

Applications use the getName method on the UserWorkArea interface to retrieve the name of the current work area. This is the recommended method for determining whether the thread is associated with a work area; if the thread is not associated with a work area, the getName method returns null. In the following code example, the name of the work area corresponds to the name of the class in which the work area was begun.

```
public class SimpleSampleBeanImpl implements SessionBean {

    ...

    public String [] test() {
       // Get the work-area reference from JNDI.
       ...

       // Retrieve the name of the work area. In this example,
       // the name is used to identify the class in which the
       // work area was begun.
       String invoker = userWorkArea.getName();
       ...
   }
}
```

Related tasks

"Overriding work area properties"

"Retrieving work area properties" on page 1523

"Retrieving a list of all keys in a work area" on page 1523

"Querying the mode of a work area property" on page 1524

"Deleting a work area property" on page 1524

Overriding work area properties:

Work areas are inherently associated with the process that creates them. In the sample application, the client begins a work area and sets into it the site-identifier and priority properties. This work area is propagated to the server when the client makes a remote invocation.

Applications nest work areas in order to temporarily override properties imported from a client process. The nesting mechanism is automatic; invoking begin on the UserWorkArea interface from within the scope of an existing work area creates a nested work area that inherits the properties from the enclosing work area. Properties set into the nested work area are strictly associated with the process in which the work area was begun; the nested work area must be completed within the process that created them. If a work area is not completed by the creating process, the work-area facility terminates the work area when the process exits. After a nested work area is completed, the original view of the enclosing work area is restored. However, the view of the complete set of work areas associated with a thread cannot be decomposed by downstream processes.

Applications set properties into a work area using property modes in ensure that a particular property is fixed (not removable) or read-only (not overrideable) within the scope of the given work area.

In the following code example, the server-side sample bean attempts to write directly to the imported work area; because the UserWorkArea partition is not defined to be bidirectional, this action is not permitted, and the NotOriginator exception is thrown. When the UserWorkArea partition is not defined as bidirectional, the sample bean must begin its own work area in order to override any imported properties, as shown in the second code example. If a work area in a user defined partition is used and is defined as bidirectional, this bean can set context into the work area before beginning another work area. This context set in the bidirectional case propagates back to the caller. See “Bidirectional propagation” on page 1534for additional information.

```
public class SimpleSampleBeanImpl implements SessionBean {

   public String [] test() {
     ...
     String invoker = userWorkArea.getName();

     try {
       userWorkArea.set("key", "value");
     }
     catch (NotOriginator e) {
     }
     ...
  }
}
```

The following code example demonstrates beginning a nested work area, using the name of the creating class to identify the nested work area.

```
public class SimpleSampleBeanImpl implements SessionBean {

   public String [] test() {
      ...
      String invoker = userWorkArea.getName();
      try {
        userWorkArea.set("key", "value");
      }
      catch (NotOriginator e) {
      }

      // Begin a nested work area. By using the name of the creating
      // class as the name of the work area, we can avoid having
      // to explicitly set the name of the creating class in
      // the work area.
      userWorkArea.begin("SimpleSampleBean");

      ...
   }
}
```

In the example application, the client sets the site-identifier property as read-only; that guarantees that the request is always associated with the client’s company identity. A server cannot override that value in a nested work area. In the following code example, the SimpleSampleBean attempts to change the value of the site-identifier property in the nested work area it created.

```
public class SimpleSampleBeanImpl implements SessionBean {

  public String [] test() {
      ...

      String invoker = userWorkArea.getName();
      try {
        userWorkArea.set("key", "value");
      }
      catch (NotOriginator e) {
      }

      // Begin a nested work area.
```

```
      userWorkArea.begin("SimpleSampleBean");

      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }
      ...
  }
}
```

Related reference

"Example: WorkArea SimpleSample application" on page 1517

"Bidirectional propagation" on page 1534

Related information

"Nested work areas" on page 1511

"Work area property modes" on page 1510

"Setting properties in a work area" on page 1519

Retrieving work area properties:

Properties can be retrieved from a work area by using the get method. This method is intentionally lightweight; there are no declared exceptions to handle. If there is no active work area, or if there is no such property set in the current work area, the get method returns null.

Note: The get method can raise a NotSerializableError in the relatively rare scenario in which CORBA clients set composed data types and invoke enterprise-bean interfaces.

The following example shows the retrieval of the site-identifier and priority properties by the SimpleSampleBean. Recall that one property was set into an outer work area by the client, and the other property was set into the nested work area by the server-side bean; the nesting is transparent to the retrieval of the properties.

```
public class SimpleSampleBeanImpl implements SessionBean {

    public String [] test() {
      ...

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");
      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }

      SimpleSampleCompany company =
         (SimpleSampleCompany) userWorkArea.get("company");
      SimpleSamplePriority priority =
         (SimpleSamplePriority) userWorkArea.get("priority");
       ...
   }
}
```

Related concepts

"Work area property modes" on page 1510

"Nested work areas" on page 1511

Retrieving a list of all keys in a work area:

The UserWorkArea interface provides the retrieveAllKeys method for retrieving a list of all the keys visible from a work area. This method takes no arguments and returns an array of strings. This method returns null if there is no work area associated with the thread. If there is an associated work area containing no properties, the method returns an array of size 0.

Querying the mode of a work area property:

The UserWorkArea interface provides the getMode method for determining the mode of a specific property. This method takes the property's key as an argument and returns the mode as a PropertyModeType object. (See Setting property modes for more information on names of mode types.) If the specified key does not exist in the work area, the method returns PropertyModeType.normal, indicating that the property can be set and removed without error.

Related concepts

"Work area property modes" on page 1510

Deleting a work area property:

The UserWorkArea interface provides the remove method for deleting a property from the current scope of a work area. If the property was initially set in the current scope, removing it deletes the property. If the property was initially set in an enclosing work area, removing it deletes the property until the current scope is completed. When the current work area is completed, the deleted property is restored.

The remove method takes the property's key as an argument. Only properties with the modes normal and read-only can be removed. Attempting to remove a fixed property creates the PropertyFixed exception. Attempting to remove properties in work areas that originated in other processes creates the NotOriginator exception.

Related concepts

"Work area property modes" on page 1510

Related tasks

"Setting properties in a work area" on page 1519

Completing a work area:

After an application has finished using a work area, it must complete the work area by calling the complete method on the UserWorkArea interface. This terminates the association with the calling thread and destroys the work area. If the complete method is called on a nested work area, the nested work area is terminated and the parent work area becomes the current work area. If there is no work area associated with the calling thread, a NoWorkArea exception is created.

Every work area must be completed, and work areas can be completed only by the originating process. For example, if a server attempts to call the complete method on a work area that originated in a client, a NotOriginator exception is created. Work areas created in a server process are never propagated back to an invoking client process.

Note: The work area service claims full local-remote transparency. Even if two beans happen to be deployed in the same server, and therefore the same JVM and process, a work area begun on an invocation from another is completed and the bean in which the request originated is always in the same state after any remote call.

The following code example shows the completion of the work area created in the client application.

```
public class SimpleSampleServlet {
   ...
   userWorkArea.begin("SimpleSampleServlet");
   userWorkArea.set("company",
       SimpleSampleCompany.Main, PropertyModeType.read_only);
```

```
   userWorkArea.set("priority", SimpleSamplePriority.Silver);
   ...

   // Do application work.
   ...

   // Terminate the work area.
   try {
      userWorkArea.complete();
   }

   catch (NoWorkArea e) {
      // There is no work area associated with this thread.
      ...
   }

   catch (NotOriginator e) {
      // The work area was imported into this process.
      ...
   }
  ...
}
```

The following code example shows the sample application completing the nested work area it created earlier in the remote invocation.

```
public class SimpleSampleBeanImpl implements SessionBean {

    public String [] test() {
      ...

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");
      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }

      SimpleSampleCompany company =
         (SimpleSampleCompany) userWorkArea.get("company");
      SimpleSamplePriority priority =
         (SimpleSamplePriority) userWorkArea.get("priority");

      // Complete all nested work areas before returning.
      try {
        userWorkArea.complete();
      }
      catch (NoWorkArea e) {
      }
      catch (NotOriginator e) {
      }
   }
}
```

Related concepts

“Nested work areas” on page 1511

Related tasks

“Beginning a new work area” on page 1518

Managing the work area service

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.

Related concepts

"Work area service - Overview" on page 1510

Related tasks

"Developing applications that use work areas" on page 1516

Enabling the work area service:

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 the service is enabled by default.

1. Enable (or disable) the use of work areas 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.
2. Enable (or disable) the use of work areas 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 `$WAS_HOME/bin` directory, add the following system property to the launchClient invocation:

   ```
   -CCDcom.ibm.websphere.workarea.enabled=false
   ```
3. 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.

Related tasks

"Managing the size of work areas" on page 1527

Related information

"WorkArea service: Special considerations" on page 1514

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.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Business process services> Work area service** .

Related tasks

"Enabling the work area service"

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 to no limit

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 to no limit

The following values are also used to define the maximum receive size.

-1	Default.
0	No limit.

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. 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 > Work area service**.
   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.
2. 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 do this in several ways. For example, to set the maximum size to 10,000 bytes, when invoking the launchClient script found in the `$WAS_HOME/bin` directory, add the following system property to the launchClient invocation:

```
-CCDcom.ibm.websphere.workarea.maxSendSize=10000
```

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.

Related reference

"Work area service settings" on page 1526
Use this page to manage the work area service.

Configuring work area partitions on the server

The work area partition service extends the work area service by allowing the creation of multiple work areas with more configuration options. Follow these steps to create and configure a work area partition:

1. Start the administrative console.
2. Click **Servers > Application servers >***server_name* **> Business process services> Work area partition service**.
3. Click **New**.
4. On the settings page for work area partitions, specify values such as the partition name, maxSendSize and maxReceiveSize, then click OK.
5. Save the new configuration and restart the server to apply the new configuration

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 work area service interface. See the topic, "Example: Work area partition manager" on page 1531, for an example.

Related reference

"Work area partition settings" on page 1533
Use this page to modify the work area service settings.

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. 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 1534 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 demarshaled 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.

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 {

  public UserWorkArea getWorkAreaPartition(String partitionName) throws NoSuchPartitionException;

   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 the remote interface; 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. 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 process at any time.

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.

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 1528 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;
    }
}
```

Related concepts

“Work area partition service” on page 1528
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.

Related tasks

“Configuring work area partitions on the server” on page 1528

"Developing applications that use work areas" on page 1516

Related reference

"The Work area partition manager interface" on page 1530

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)

Maximum receive size:

Specifies the maximum size of data that can be received within a single work area. (0 = no limit; -1 = default)

Deferred attribute serialization:

Specifies whether attribute serialization is deferred until the work area is propagated on a remote invocation.

Work area partition settings:

Use this page to modify the work area service 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_partion_name*.

Related reference

"Work area partition collection"
Use this page to manage the work area 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)

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.

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.

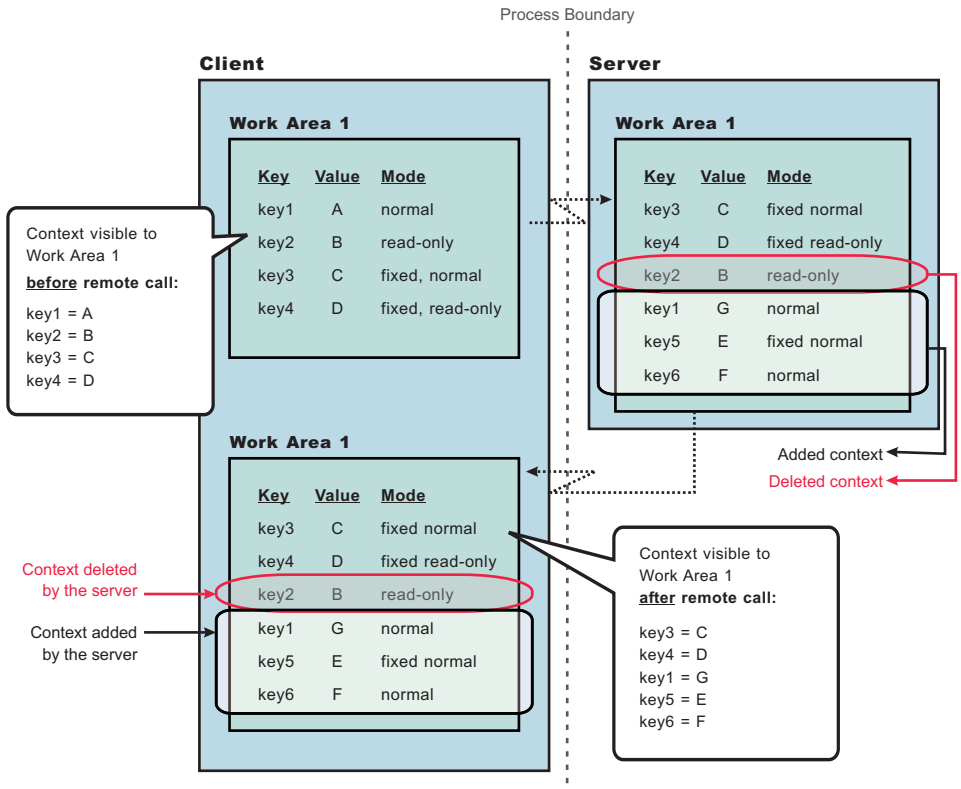


Figure 20. Figure 1

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.

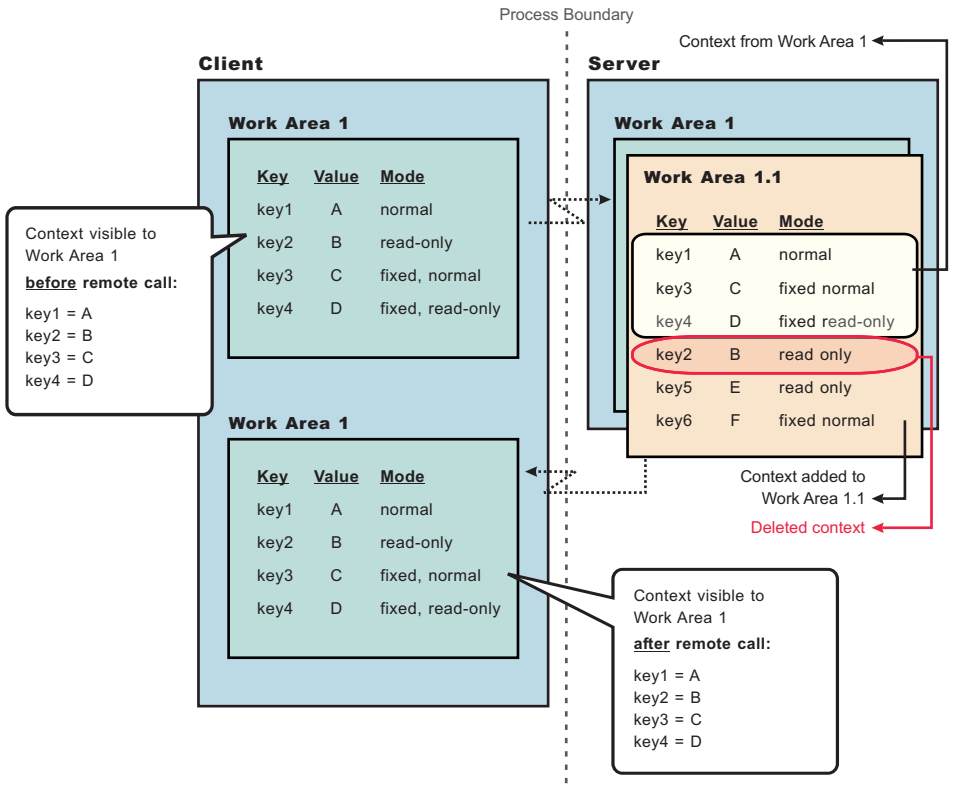


Figure 21. Figure 2

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.

Related concepts

“Work area service - Overview” on page 1510

“Work area partition service” on page 1528
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.

Related tasks

“Configuring work area partitions on the server” on page 1528

Related reference

“The Work area partition manager interface” on page 1530

Chapter 7. 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 and databases
- 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, see ″Obtaining 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 IBM Support page.

Errors or problems deploying, installing, or promoting applications

This article 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?
- I installed my application using the wsadmin tool, but the application does not display under Applications > Enterprise Applications.
- I get a `java.lang.RuntimeException: Failed_saving_bytes_to_wor_ERROR_` in the assembly tool, administrative console or wsadmin tool
- I get a WASX7015E error running the wsadmin command `$AdminApp installInteractive` or `$AdminApp install`..
- A data definition language (DDL) generated by an assembly tool throws an SQL error on the target platform.
- The error `ADMA0004E: Validation error in task Specifying the Default Datasource for EJB Modules` occurs when installing application in administrative console or the wsadmin tool.
- The error `No valid target is specified in ObjectName` *`object`* `for module` *`module`* occurs from installation.
- The addNode -includeapps option does not appear to upload all applications to the deployment manager.
- ″Timeout!!!″ error displays when attempting to install an enterprise application in the administrative console.
- I get a NameNotFoundException message when deploying an application that contains an EJB module
- During application installation, the call to EJB deploy throws an exception
- I get compilation errors and EJB deploy fails when installing an EJB JAR file generated for Version 5.x or earlier
- While uploading documents, addNode -includeapps fails with an OutOfMemoryError exception

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 ″Diagnosing and fixing problems: Resources for learning″ and Obtaining 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 *install_dir*/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 $AdminConfigsave command in the wsadmin tool before exiting.

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 EJBDeploy 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 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 "Documents and Settings/myUserName/Desktop/MyApps/myapp.ear"
```

- If the full text of the error is similar to:

```
WASX7015E: Exception running command: "$AdminApp installInteractive MyApps\myapp.ear ";
exception information: com.ibm.ws.scripting.ScriptingException:  WASX7115E:
Cannot read input file
"WebSphere\AppServer\bin\MyAppsmyapp.ear"
```

  The application path is incorrectly specified. In this case, you must use UNIX-style ″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 IDs 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 Windows operating system on a UNIX platform, 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 UNIX platforms 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 Version 5, 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, either:

- Use the **`$AdminApp installInteractive`** ***`filename`*** command to receive prompts for data sources during installation, or to provide them in a response file.

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.

addNode -includeapps option does not appear to upload all applications to the Deployment Manager

This error can occur when some or all applications on the target node are already uploaded to the deployment manager. The addNode program detects which applications are already installed and does not upload them again.

Use the administrative console to browse the deployment manager configuration and see the applications that are already installed.

"Timeout!!!" error displays when attempting to install an enterprise application in the administrative console

This error can occur if you attempt to install an enterprise application that has not been deployed.

To correct this problem:
- Open the *file_name*.ear file in an assembly tool and then click **Deploy**. This action creates a file with a name like Deployed_*file_name*.ear.
- In the administrative console, install the deployed .ear file.

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 EJB deploy 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.

While uploading documents, addNode -includeapps fails with an OutOfMemoryError exception

This error can occur when you use addNode -includeapps while you are installing applications with large EAR files. To correct this problem:

- If you are using addNode to add a node from the base server, modify the addNode script to include the following parameter:

  -Xmx*size*
- If you are adding a node from the administrative console, increase the *maximumHeapSize* in the Java virtual machine settings of the Deployment Manager, then restart the Deployment Manager. See Java virtual machine settings for details.

For example, the addNode.bat file that follows sets a maximum heap size of 512 MB on a Windows platform:

```
"%JAVA_HOME%\bin\java" -Xmx512m %DEBUG% %WAS_TRACE% %CONSOLE_ENCODING%
"%CLIENTSOAP%" "%CLIENTSAS%" "-classpath" "%WAS_CLASSPATH%"
"-Dws.ext.dirs=%WAS_EXT_DIRS%" %USER_INSTALL_PROP%
-Dwas.install.root=%WAS_HOME%" "com.ibm.ws.bootstrap.WSLauncher"
"com.ibm.ws.management.tools.NodeFederationUtility" "%CONFIG_ROOT%" "%WAS_CELL%"
"%WAS_NODE%" %*
```

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:

- “A web resource does not display” on page 1547.
- ″Cannot access a data source″.
- ″Cannot access an enterprise bean from a servlet, a JSP file, a stand-alone program, or another client″.
- ″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″.
- ″Errors returned to a client sending a SOAP request″.
- ″Errors connecting to WebSphere MQ and creating WebSphere MQ queue connection factory″.

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.

Errors 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” on page 1544
- “File serving problems” on page 1544
- “Graphics do not appear in the JSP file or servlet output” on page 1545
- “SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file” on page 1545
- “Message like "Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing" appears when attempting to browse JSP file” on page 1546

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 *install_dir*/config/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. You might need 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 and 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 *install_dir*/config/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.

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 *installation_root*/installedApps/*nodename*/*appname*.ear directory or *installation_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. Edit the **fileServingEnabled** property in the deployed Web application ibm-web-ext.xmi configuration file, typically found in the *install_root*/config/cells/*nodename* or *nodename*Network/applications/*application name*/deployments/*application name*/*Webmodule 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 *installation_root*/installedApps/*nodename*/*appname*.ear directory or *installation_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. Edit the **fileServingEnabled** property in the deployed Web application `ibm-web-ext.xmi` configuration file, typically found in the *install_root*/config/cells/*nodename* or *nodename*Network/applications/*application name*/deployments/*application name*/*Webmodule name*/web-inf directory.
  2. 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 thejavac compilation phase.

Check the log files for a compiler error message, such as:

```
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 servlets served by class name, the URL is `http://`*`hostname/Web_module_context_root`*`/servlet/`*`packageName.className`*.

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`)

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 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 *`install_root`*`\config\cells\`*`node_name`*`Network\applications` directory is expected as an argument to the JspBatchCompiler tool, not just the display name. 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() 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**.

Related reference

"A web resource does not display"

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 logs 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`
   - For servlets or other resources served by mapped URLs, the URL is http://*hostname*/*web module context root*/*mappedURL*.

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.
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″
- The `removeNode` command does not remove the installed application from the deployment manager
- I cannot display the syntax for the `removeNode` command.

If none of these steps fixes your problem:
- Make sure that the application and its Web and EJB modules, are in a stopped state before uninstalling.
- If you are uninstalling or installing an application using **wsadmin**, make sure that you are using the **-conntype NONE** option to invoke **wsadmin** and enable local mode. To use the **-conntype NONE** option, stop the hosting application server before uninstalling the application.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don’t find your problem listed there contact IBM support

After uninstalling application through the wsadmin tool, the application throws "DocumentIOException"

If this exception occurs after the application was uninstalled using wsadmin with the `-conntype NONE` option:
- Restart the server or,
- Rerun the uninstall command without the `-conntype NONE` option.

The removeNode command does not remove the installed application from the deployment manager

If the applications were installed indirectly using the **addNode** program with the **-includeapps** option, then `removeNode` will not uninstall them, since they may be in use by other nodes. These applications must be explicitly uninstalled, for example through the administrative console.

I cannot display the syntax for the removeNode command

Unlike the `addNode` command, the `removeNode` command is valid with no parameters, so executing it will execute the operation, that is, remove the node, without displaying the command syntax.

To see the valid options for removeNode, execute `removeNode -?` or `removeNode -help`.

Chapter 8. Troubleshooting administration

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.
- My ″Web module or application server dies or hangs″.
- I get ″errors trying to configure and enable security″.
- 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 Netscape browser.

Administration and administrative console troubleshooting tips

In WebSphere Application Server products, administrative functions are supported by:
- The application server (such as server1) process
- The deployment manager (dmgr) process in the Network Deployment product

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.

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. To fix this, increase the timeout value specified by the `com.ibm.SOAP.requestTimeout` property in the `soap.client.props` file in the *install_root*`/profiles/`*profile_name*`/properties` directory for a single server edition or in the *install_root*`/profiles/`*profile_name*`/properties` directory for a network deployment installation. The value you should 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.

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:
  - *install_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*.**.
  - the server log files for the message that indicates that the server started successfully: **WSVR0001I: Server *server* open for e-business**.
- For the Network Deployment product, look at these files:
  - *install_root*`/profiles/`*profile_name*`/logs/dmgr/startServer.log` for the message that indicates that the server started successfully: **ADMU3000I: Server dmgr open for e-business; process id is *nnnn*.**.
  - the server log files for this message indicating that the server started successfully:**WSVR0001I: Server dmgr open for e-business**.
- For the z/OS product, check the job output.
- Look up any error messages in these files in the message reference table. Select the **Reference** view in the information center navigation, then 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 used by wsadmin is incorrect.
- Verify that you are using the right port number to communicate with the administrative console or wsadmin server using the following steps:
  - Look in the `joblogs` file.
  - The line **ADMC0013I: SOAP connector available at port *nnnn*** indicates the port that the server is using to listen for wsadmin functions.
  - The property **com.ibm.ws.scripting.port** in the *install_root*/profiles/*profile_name*/properties/wsadmin.properties file controls the port used by wsadmin to send requests to the server. If it is different from the value shown in the the server log files, either change the port number in the `wsadmin.properties` file, or specify the correct port number when starting wsadmin by using the **-port *port_number*** property on the command line.
  - The message **SRVE0171I: Transport http is listening on port *nnnn* (default 9060)** indicates the port the server uses to listen for administrative console requests. If it is different than the one specified in the URL for the administrative console, change the URL 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 or deployment manager is running, is reachable from the system where the browser or wsadmin program are being used. If you are able to ping the host name, this indicates that there are no firewall or connectivity issues.
- If the host where the application server or deployment manager 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:
  - The host name in the browser URL for the console.
  - The **-host *host name*** option of the wsadmin command that is used to direct wsadmin to the right server

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 information 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

Administrative console problems

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 console page.
- The directory paths in the console contain strange characters.

If you can bring up the browser page, but the console behavior is inconsistent, error prone, or unresponsive, try upgrading your browser. Older browsers might not support all the features of the administrative console. For a listing of supported Web browsers, see http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html.

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:
- If you are using the deployment manager for a multi-node configuration, run the **startManager** command from the *Network_Deployment_install_dir*\bin directory.
- View the file for the application server or the deployment manager to verify that the server supporting the administrative console has started.
- Verify that the application server which 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 verify that server1 is running. Do so by issuing the following command on the MVS console to list active processes:

    ```
    D A,L
    ```

    Note: See z/OS MVS System Commands for information on how to use MVS operator commands. Check for the application server procedure name of BB05ACR with the server short name of BB0S001. If it is not running, enter the following command on the MVS console:

    ```
    START appserver_proc_name,JOBNAME=server_short_name,
          ENV=cell_short_name.node_short_name.server_short_name
    ```

    Example:

    ```
    START BBO5ACR,JOBNAME=BBOS001,ENV=PLEX1.SY1.BBOS001
    ```
  - For a network deployment configuration, the deployment manager runs the administrative console. For example, to start the deployment manager, you can issue the following command from the MVS console:

    ```
    START BBO5DCR,JOBNAME=BBODMGR,ENV=PLEX1.PLEX1.BBODMGR
    ```
  - View the joblog or sysprint for the application server or deployment manager to verify that the server supporting the administrative console has started.
- Check the URL you use to view the console.
- Try to eliminate connection, address and firewall issues by pinging the server machine from a command prompt, using the server name in the URL.

An "Unable to process login. Check user ID and password and try again. " error occurs when trying to access console page

This error indicates that security is enabled for WebSphere Application Server, and that the user ID or password supplied is either invalid 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 file security.xml.

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 http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html.

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

If you are having problems using a Web server plug-in, try these steps:
- Review the file *plugin_install_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 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

    The GSKit did not get installed or the installation is corrupt. If the GSKit did not get installed you can determine this by searching for the file gsk7ssl.dll on all drives for Win32 or see if there are any libgsk7*.so files in /usr/lib on UNIX. Try reinstalling the plug-in to see if you can get the GSKit to install in order to fix this.

    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 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 you are trying to use with the Web server plug-in in the information center topic, ″Communicating with Web servers″.
- 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 can't be pinged then there is no way the plug-in will be able to contact them. Possible reasons for this include:
  - Firewall policies limiting the traffic from the plug-in to the app server.
  - The machines are not on the same network.
– If you are able to ping the machines then the likely cause of the problem is that the port is not active. This could be because the application server or cluster has not been started or the application server has gone down for some reason. You can test this by hand by trying to 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 will be able to connect successfully.

- Determine whether other activity on the machines where the servers are installed is impairing the server's ability 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 to ensure that the application servers are started. View the administrative console for error messages.
- 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.

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

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the following topics on the IBM support page:
- HTTP transport
- HTTP plug-in
- HTTP plug-in remote install

Cannot restart the Deployment Manager monitoring policy

The Deployment Manager monitoring policy is permanently set to STOPPED and cannot be changed. Therefore, if the Deployment Manager fails it will never be restarted by WebSphere Application Server monitoring.

To correct the problem, use the z/OS Automatic Restart Facility (ARM) to monitor and restart the Deployment Manager. See z/OS Automatic Restart Facility for information on ARM.

Errors setting up multiserver environments

What kind of problem are you seeing?

- "After creating and starting a cluster, the cluster does not start, and logs show that servers in the cluster are not found"
- "One or more nodes do not show up in the administrative console" on page 1558
- "The addNode command fails" on page 1558
- "Application files are not present on all nodes" on page 1558
- "After downloading the Network Deployment plug-in to my system, my server does not start" on page 1559
- "In a clustered environment, a server with debug mode enabled does not start" on page 1559

If none of these problem solution descriptions fixes your problem:

1. Browse the logs of the problem deployment manager and application servers:
   a. Look up any error messages by selecting the **Reference** view of the information center navigation and expanding **Messages** in the navigation tree.
   b. If Java exceptions appear in the log files, try to determine the actual subcomponent directly involved in the problem by examining the trace stack and looking for a WebSphere Application Server-related class near the top of the stack (names beginning with `com.ibm.websphere` or `com.ibm.ws`) that threw the exception. If appropriate, review the steps for troubleshooting the appropriate subcomponent in the ″Troubleshooting by component″ topic in the information center.

      For example, if the exception appears to have been thrown by a class in the `com.ibm.websphere.naming` package, review the ″Naming services component troubleshooting tips″ topic.
2. Ensure that all the machines in your configuration have TCP/IP connectivity to each other by running the **ping** command:
   a. From each physical server to the Deployment Manager
   b. From the Deployment Manager to each physical server
3. Although the problem is happening in a clustered environment, the actual cause might be only indirectly related, or unrelated, to clustering. Investigate all relevant possibilities:
   a. If an enterprise bean on one or more servers is not serving requests, review the Cannot access an enterprise bean from a servlet, JSP, stand-alone program, or other client and Cannot access an object hosted by WebSphere Application Server from a servlet, JSP file, or other client topics.
   b. If problems seem to appear after enabling security, review the Errors or access problems after enabling security topic.
   c. If an application server stops responding to requests, or spontaneously dies (its process closes), review the Web module or application server dies or hangs topic.
   d. If SOAP requests are not being served by some or all servers, review the Errors returned to client trying to send a SOAP request topic.
   e. If you have problems installing or deploying an application on servers on one or more nodes, review the Errors or problems deploying, installing, or promoting applications topic.
4. If your topology consists of a Windows-based Deployment Manager with UNIX-based servers, browse any recently-updated `.xml` and `.policy` files on the UNIX-based platform using **vi** to ensure that Control-M characters are not present in the files. Edit these files using **vi** on the UNIX-based platform, to avoid inserting these characters.
5. Check the steps for troubleshooting the Workload Management component..
6. Check to see if the problem is identified and documented by looking at available online support (hints and tips, technotes, and fixes).

After creating and starting a cluster, the cluster does not start, and logs show that servers in the cluster are not found

This error can occur when the configuration is not synchronized from the deployment manager to a node. If ***auto synchronization*** is enabled, wait until the synchronization has had a chance to run. If you are using manual synchronization, explicitly request a sync to each node on the cluster.

To determine whether synchronization has taken place, look at the configuration on the node machines using the administrative console and verify that the new cluster members are defined on each node.

One or more nodes do not show up in the administrative console

This can occur when there is a basic connectivity problem between the deployment manager server and other servers in the topology. To determine whether this is the problem, look for the file`serverindex.xml` in the deployment manager directory structure.

- If the problem node does not appear in the list, review the steps for adding a node to the cluster.
- If the problem node does appear in the list:
  - From the deployment manager server, ping the server name as it appears in the list. If the ping command shows no communication, verify that the host name is correct in the list, and correct it if necessary, then restart the deployment manager.
  - If the name that appears in the list is the short name, ping the fully qualified network name. If the corrected name works, update the list and restart the deployment manager.
  - If the problem server uses Dynamic Host Configuration Protocol (DHCP), try replacing the logical host name with the IP address and restart the deployment manager. If this resolves the problem, be aware that you must change `serverindex.xml` each time the problem server address changes, potentially each time the problem machine is rebooted. To avoid this problem, consider assigning a static IP address to the server.
- If you still cannot establish communication between the servers, contact your network administrator to resolve the problem, and restart the deployment manager after the problem is corrected.

The addNode command fails

This error can occur when the deployment manager Domain Name Server (DNS) configuration is set up improperly. The default installation on Linux uses the loopback address (127.0.0.1) as the default host address. To verify that this is the problem, query the host name of the suspect machine. If it returns localhost 127.0.0.1, or if file transfer traces at the node show the node trying to upload files to a URL that includes 127.0.0.1, the node has an incorrect DNS configuration.

To correct this problem, update the `/etc/hosts` file or the name service configuration file, `/etc/nsswitch.conf`, to query the Domain Name Server or Network Information Server (NIS) before searching hosts.

Application files are not present on all nodes

In the WebSphere Application Server Network Deployment environment, application binary files are transferred to the individual nodes where applications are supported as part of the **node sync** operation. During node sync, application files are only propagated if their deployment descriptors specify **enableDistribution=true**. This flag is specified as part of the application installation procedure in the administrative console, and is stored as a property in the *install_root*`/config/cells/`*cell_name*`/applications/`*application_name*`/deployment.xml` file.

To confirm that this is the cause, check to see whether the enableDistribution flag is set. If it is already set to true, ensure that the target node is configured to run auto file synchronization.

If both of these settings are correct and the problem persists, manually perform an explicit synchronization. If the application files still do not appear in the installation directory, use the EARExpander tool (located in *install_root*`/bin`) to expand the EAR file from the repository to the installation destination. On remote nodes, the repository should appear in the`config/cells/`*cell_name*`/applications/`*application_name*`.ear/` directory.

After downloading the Network Deployment plug-in to my system, my server does not start

If you experience this situation, the most likely cause is that the transport paths in the plug-in must be modified to work in your environment. See the ″HTTP transport custom properties″ topic for information on how to modify these settings.

In a clustered environment, a server with debug mode enabled does not start

This problem occurs when the following three conditions exist:

- Multiple server processes are configured to run on the same node
- More than one server has Debug Mode enabled
- The debug arguments for more than one of the servers have been left at the default values, so that more than one server in the node is trying to use the same debug port (port number 7777).

The server will not start because multiple servers processes running on the same physical host machine with debug enabled cannot use the same debug port.

To correct this problem, for each server:

1. On the Administrative Console select **Server > Application servers >** *server_name* **> Java and Process Management > Process Definition > Java Virtual Machine**
2. Update the Debug argument so that the address of the debug port (address=*port number*) is unique for each server process.

Workload management component troubleshooting tips

If the workload management component is not properly distributing the workload across servers in multi-node configuration, use the following options to isolate the problem.

- Ensure that the workload is distributed across clustered servers
- Resolve any problems with the multiserver Deployment Manager environment setup
- “Eliminate environment or configuration issues”
- “Resolve problem or contact IBM support” on page 1560

Eliminate environment or configuration issues

Determine if the servers are capable of serving the applications for which they have been enabled. Identify the cluster that has the problem.

- Are there network connection problems with the members of the cluster or the administrative servers, for example deployment manager or node agents?
  - If so, `ping` the machines to ensure that they are properly connected to the network.
- Is there other activity on the machines where the servers are installed that is impacting the servers ability to service a request? For example, check the processor utilization as measured by the task manager, processor ID, or some other outside tool to see if:
  - It is not what is expected, or is erratic rather than constant.
  - It shows that a newly added, installed, or upgraded member of the cluster is not being utilized.
- Are all of the application servers you started on each node running, or are some stopped?
- Are the applications installed and operating?
- If the problem relates to distributing workload across container-managed persistence (CMP) or bean-managed persistence (BMP) enterprise beans, have you configured the supporting “JDBC providers” on page 512 and “Data sources” on page 514 on each server? For problems relating to data access, review the topic Cannot access a data source.

If you are experiencing workload management problems related to HTTP requests, such as HTTP requests not being served by all members of the cluster, be aware that the HTTP plug-in balances the load across all servers that are defined in the PrimaryServers list if affinity has not been established. If you

do not have a PrimaryServers list defined then the plug-in load balances across all servers that are defined in the cluster if affinity has not been established. If affinity has been established, the plug-in should go directly to that server for all requests.

For workload management problems relating to enterprise bean requests, such as enterprise bean requests not getting served by all members of a cluster:
- Are the weights set to the allowed values?
  - For the cluster in question, log onto the administrative console and:
    1. Select **Servers > Clusters**.
    2. Select your cluster from the list.
    3. Select **Cluster members**.
    4. For each server in the cluster, click on *server_name* and note the assigned weight of the server.
  - Ensure that the weights are within the valid range of 0-20. If a server has a weight of 0, no requests are routed to it. Weights greater than 20 are treated as 0.

Note: The remainder of this article deals with enterprise bean workload balancing only. For more help on diagnosing problems in distributing Web (HTTP) requests, view the topics “Web server plug-in troubleshooting tips” on page 1555 and “A web resource does not display” on page 1547.

Resolve problem or contact IBM support

If the client logs indicate an error in WLM, collect the following information and contact IBM support.
- A detailed description of your environment.
- A description of the symptoms.
- The server log files for all servers in the cluster.
- A description of what the client is attempting to do, and a description of the client. For example, 1 thread, multiple threads, servlet, J2EE client, etc..

If none of these steps solves the problem, 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 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.

Workload is not getting distributed

What kind of problem are you seeing?
- “Web (HTTP) requests are not distributed to all servers” on page 1561
- “Enterprise bean requests are not distributed to all servers” on page 1561
- “A failing server still receives enterprise bean requests (failover is not completed)” on page 1561
- “Workload is not getting distributed”
- “A cluster does not fail over to its backup cluster” on page 1562

If none of these problem solution descriptions fix your problem:
1. Browse the JVM logs of the problem deployment manager and application servers:
   a. Look up any error messages by selecting the **Reference** view of the information center navigation and expanding **Messages** in the navigation tree.
   b. If Java exceptions appear in the log files, try to determine the actual subcomponent that is directly involved in the problem by examining the trace stack and looking for a WebSphere Application Server-related class near the top of the stack (names beginning with `com.ibm.websphere` or

com.ibm.ws) that created the exception. If appropriate, review the steps for troubleshooting the appropriate subcomponent under the Troubleshooting by component: what is not working? topic.

For example, if the exception appears to have been thrown by a class in the com.ibm.websphere.naming package, review the Naming Services Component troubleshooting tips topic.

2. Ensure that all the machines in your configuration have TCP/IP connectivity to each other by running the **ping** command:
   a. From each physical server to the deployment manager
   b. From the deployment manager to each physical server
3. Although the problem is happening in a clustered environment, the actual cause might be only indirectly related, or unrelated, to clustering. Investigate all relevant possibilities:
   a. If an enterprise bean on one or more servers is not serving requests, review the Cannot access an enterprise bean from a servlet, JSP, stand-alone program, or other client and Cannot access an object hosted by WebSphere Application Server from a servlet, JSP file, or other client topics.
   b. If problems seem to appear after enabling security, review the Errors or access problems after enabling security topic.
   c. If an application server stops responding to requests, or spontaneously dies (its process closes), review the Web module or application server dies or hangs topic.
   d. If SOAP requests are not being served by some or all servers, review the Errors returned to client trying to send a SOAP request topic.
4. Check to see if the problem is identified and documented by looking at available online support (hints and tips, technotes, and fixes).

Web (HTTP) requests are not distributed to all servers

If HTTP requests are not being distributed to all servers:

- Check your PrimaryServers list. The plug-in load balances across all servers that are defined in the PrimaryServers list, if affinity has not been established. If you do not have a PrimaryServers list defined, the plug-in load balances across all servers defined in the cluster, if affinity has not been established. In the case where affinity has been established, the plug-in should go directly to that server, for all requests within the same HTTP session.
- If some servers are servicing requests and one or more others are not, try accessing a problem server directly to verify that it works, apart from workload management issues. If that does not work:
  – Use the administrative console to ensure that the affected server is running.
  – See the article “A web resource does not display” on page 1547 for more information.
- See the article HTTP plug-in component troubleshooting tips for more information.

Enterprise bean requests are not distributed to all servers

If a client cannot reach a server in a cluster thought to be reachable, a server might be marked unusable, or is down. To verify this:

- Use the administrative console to verify that the server is started. Try starting it, or if started, stop and restart it.
- Browse the administrative console and verify that the node that runs the server having the problem appears. If it does not:
  – Review the steps for adding a node to a cluster.
  – Review the steps in the One or more nodes do not show up in the administrative console topic.
- If possible, try accessing the enterprise bean directly on the problem server to see if there is a problem with TCP/IP connectivity, application server health, or other problem not related to workload management. If this fails, review the topic Cannot access enterprise bean from a servlet, JSP, stand-alone program , or other client.

A failing server still receives enterprise bean requests (failover is not completed)

Some possible causes of this problem are:

- The client might have been in a transaction with an enterprise bean on the server that went down. Check the JVM logs of the application server hosting the problem enterprise bean instance. If a request is returned with **CORBA SystemException COMM_FAILURE org.omg.CORBA.completion_status.COMPLETED_MAYBE**, this might be working as designed. The design is to let this particular exception flow back to the client, since the transaction might have completed. Failing over this request to another server could result in this request being serviced twice.
- If the requests sent to the servers come back to the client with any other exceptions consistently, it might be that no servers are available.

A cluster does not fail over to its backup cluster

You might experience an error that is similar to the following sample:

```
[10/11/04 13:11:10:233 CDT] 00000036 SelectionMana A    WWLM0061W: An error was
encountered sending a request to cluster member  {MEMBERNAME=FlorenceEJBServer1,
NODENAME=fwwsaix1Node01} and that member has been  marked unusable for future
requests to the cluster "", because of exception:  org.omg.CORBA.COMM_FAILURE:
CONNECT_FAILURE_ON_SSL_CLIENT_SOCKET - JSSL0130E:  java.io.IOException: Signals
that an I/O exception of some sort has occurred.   Reason:  Connection refused
vmcid: 0x49421000  minor code: 70  completed: No"
```

Perform the following steps to fix your configuration:

1. Review your deployment manager hostname and bootstrap port for each backup cluster setting.
2. Review your core group bridge peer ports to make sure the hostname and DCS port are accurate.
3. Verify that the names of your primary and backup clusters match.
4. If your application is going through security to go to the backup cluster, review your security configuration. You might need to use single sign on (SSO) and import the Lightweight Third Party Authentication (LTPA) keys to the backup cell.

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 starting or using the wsadmin command

What kind of problem are you having?

- ″WASX7023E: Error creating ″SOAP″ connection to host″ or similar error trying to launch wsadmin command line utility.
- ″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.
- WASX7022E returned from running ″wsadmin -c ...″ command, indicating invalid command.
- com.ibm.ws.scripting.ScriptingException: WASX7025E: String ″″ is malformed; cannot create ObjectName.
- WASX701E: Exception received while running file ″*scriptName*.jacl″; exception information: com.ibm.bsf.BSFException: error while evaluating Jacl expression: missing close-bracket

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.

"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.sh** *server short name* 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 z/OS configuration, you will first need to start the deployment manager by issuing the following command from a command prompt on the MVS console:

  ```
  START dmgr_proc_name,JOBNAME=server_short_name,
        ENV=cell_short_name.node_short_name.server_short_name
  ```

  Note: This command must be entered on a single line. It is split here for display purposes.

  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 *install_dir*/profiles/*profile_name*/config/cells/*node_name*/nodes/*node_name*/serverindex.xml 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 *install_dir*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/serverindex.xml 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 shell attempting to do variable substitution. Variable substitution can occur on Unix System Services if wsadmin -c invokes a command that is enclosed in double quotes and includes dollar signs. 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 ...**″.

Note: When the command is enclosed in single quotes, the shell does not attempt to do variable substitution. Therefore, you do not need to escape the dollar sign. Example: **wsadmin.sh -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.

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.
-

Problems using tracing, logging or other troubleshooting features

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.

Resolving timeout conditions

This file gives an overview of how to resolve timeout conditions

In such a complex environment as WebSphere Application Server for z/OS, timeouts might occur for many different reasons. Although you can alter timeout values, you should not do so until you understand why the timeout occurs. Depending on the timeout condition, you might be able to permanently fix the timeout condition by doing some system or application tuning. For example, if the diagnostic data indicates throughput problems, you can alter the number of server regions, the number of threads within each server region, or the use of replicated servers.

Generally speaking, increasing the timeout values should be your last resort, or only a temporary action taken to prevent multiple timeout-abend dumps from causing system performance problems. If you increase timeout values without properly diagnosing the timeout condition, the only results you might see are less frequent abends and dumps for the same timeout condition, or slower system or application performance.

Understanding how timers work

This file gives an overview of understanding timers

Timers define a limit to the amount of time required for a specific operation to complete. When the timer begins its countdown depends on type of operation it controls. The timers that WebSphere Application Server for z/OS uses can be classified into the general types described later in this article; the specific timers themselves are described in ″Controlling behavior through timeout values″ in the information center.

Most of the timers have a default value that defines a reasonable limit for the particular operation to complete. When the timer pops (that is, reaches the time limit), WebSphere Application Server for z/OS takes one of the following actions:

- Sends a minor code to the client for timers that pop before the client request is dispatched to a servant region.
- Abnormally ends the servant region with an EC3 ABEND for timers that pop while the client request is being processed by an application component running in the servant region. All threads in the abending servant region will be terminated.

  WebSphere terminates the servant region to prevent the application from tying up resources, thus causing other requests to start backing up. Once the servant is terminated, WLM starts a new servant to take its place and continue processing requests from the controller.

  Note: The total transaction lifetime timeout and the maximum transaction timeout have grace periods beyond the timeout value specified of about four minutes that must be reached before an ABEND occurs.

Different types of timers might be counting down simultaneously, because the operations they control might overlap to a certain degree. For example, suppose the application server receives an IIOP client request that will be processed by an application component that uses transaction support. In this case, both of the following WebSphere timers can be counting down simultaneously:

- control_region_wlm_dispatch_timeout, which limits both the amount of time a client request waits on the WLM queue, as well as the time required for the application component to process the request; and
- transaction_defaultTimeout, which limits the amount of time the controller will wait for a transaction to be either committed or rolled back.

These timers overlap only for the time during which the application's transaction is being processed. To determine which timer cause the error, you can use the symptoms- specific minor codes or EC3 ABEND reason codes.

General types of timers and the operations they control

General type	Timer processing	Timeout symptoms
Input	Input timers define limits for receiving a complete request; the countdown starts when a connection to the J2EE server is established. The communication protocol (HTTP, HTTPS) determines the timer used for the request.	The session is closed.
Session	Session timers define limits for the use of session connections. These timers start the countdown as soon as a session becomes idle.	The session is closed.
WLM dispatch	Dispatch timers control how long a complete client request waits to be dispatched in a servant region for processing. The countdown starts when the controller places the request on the WLM queue. Depending on the specific timer, the time limit can include not only wait time on the WLM queue, but also the time required for processing a response to the client request.	Message BBOO0232W and an EC3 ABEND in the servant (region), with one of these accompanying reason codes: `04130003 04130004 04130006`

General type	Timer processing	Timeout symptoms
Transaction	Transaction timers define how long: • An application or controller will wait for one transaction to complete. The countdown starts when the container starts a transaction on behalf of the application component. • A controller will attempt to recover in-doubt transactions during peer restart and recovery mode.	• Message BBOT0003W or BBOO0232W • An EC3 ABEND in the servant (region), with one of these accompanying reason codes: `04130002 04130005`
Output	Output timers define how long a controller will wait to receive output for a client request. The countdown starts when the client request is dispatched to the servant region for processing. The communication protocol (HTTP, HTTPS) determines the timer used for the request.	Message BBOO0232W and an EC3 ABEND in the servant (region), with reason code 04130007

Guidelines for analyzing diagnostic data for timeout conditions

This file gives an overview of how to enable and use the System Management Facilities (SMF) to collect and record system and job-related information.

The following guidelines provide instructions for finding diagnostic data in an SVC dump that can help you determine what timeout condition occurred:

- Find the task with the EC3 abend:
  1. Format the TCB summary for the servant that was timed out by entering the following command:

     ```
     ip summ format asid(x' address ')
     ```

     where *address* is the address space ID of the servant.

     Find the TCB that had the EC3 completion code. Ignore any EC3 completion code on the ″main″ thread which is the 4th TCB listed in the summary format (the 1st one after the 3 MVS TCBs). The WebSphere main thread is the one that is waiting in BBO_BOA::impl_is_ready. No application requests are ever dispatched on this thread, therefore there is nothing to timeout. During timeout processing the main thread for the server region is also abended with EC3 as a mechanism of bringing the address space down. Thus the reason why the EC3 completion code may appear on the main thread. This is never the cause of a timeout though, only a result of timeout processing.
  2. If there is no EC3 completion code in the TCB summary, look in systrace. Format the systrace in GMT time since the other timestamps you'll be comparing it to are in GMT time. To format in GMT time, enter the following command::

     ```
     ip systrace all time(gmt)
     ```

     You may not see the EC3 abend in systrace either as systrace can cover a small amount of time.
  3. You can also try looking in ip verbx mtrace or in syslog to see when the EC3 abend occurred. You'll need this time to determine the 'end' time of the request which is the GMT time the timeout value was reached.
- Determine what timeout values are in effect by checking the reason code associated with the EC3 abend.

Reason code	Explanation
04130002	The controller issued an ABTERM for this servant region because a transaction timeout occurred. Code under dispatch could have been in a tight loop.
04130003	The controller issued an ABTERM for this servant region because it was hung trying to move a controller request into the servant region. The target request was timed out, but the servant was currently copying the request. The controller checked the servant for progress at regular intervals, before taking action by issuing an ABTERM.
04130004	The controller issued a ABTERM for this servant region because the WLM queue timeout occurred. Code under dispatch could have been in a tight loop.
04130005	The controller issued an ABTERM for this servant region because a transaction timeout occurred. The transaction has timed out, but no current request associated with the transaction was found. The servant associated with the transaction will be terminated.
04130006	A controller thread encountered a problem while processing a request. The request has been queued to WLM and associated with a servant region. The termination of the associated servant region is needed to complete cleanup for the request.
04130007	The controller issued a ABTERM for this servant region because the HTTP OUTPUT timeout occurred. Code under dispatch could have been in a tight loop.

- Find the method name to determine if it was

  `httpRequest`

  ,

  `httpsRequest`

  or

  `DispatchbyURI`

  or some other method.

  If the request is not specifically a request that came through the HTTP or HTTPS transport handlers, the

  `protocol_http_output_timeout`

  (HTTP) and

  `protocol_https_timeout_output`

  (HTTPS) timeout values will not be a factor. In other words, when the request is a

  `DispatchbyURI`

  method, the request is received through the RMI/IIOP protocol, so the

  `protocol_http`

  * variables have no affect.
- Obtain the callback stack for the request, using the IPCS verbexit LEDATA, with the CEEDUMP or NTHREADS option.

Identifying possible causes of and fixes for timeout conditions

This file lists common timer variables and tools for monitoring these timeout conditions

The timer that expires first might not indicate the actual problem that needs to be fixed. To properly diagnose timeout conditions, you should know all of the timer values that might be in effect for a particular servant region.

General type of timer	Possible causes	Possible solutions
Input	The client sent only part of the data and was delayed in sending the rest.	The application on the client side may want to consider having retry logic in place if it does receive a timeout minor code in return.
Session	The session is idle through lack of use.	If you consider losing idle sessions to be a problem, increase the values of the persistent-session timeouts, or use the session more frequently.
WLM dispatch	No threads are free to pick up the request because of one of the following conditions: • The threads are all busy processing requests. • The currently executing threads are waiting for a response from DB2, WebSphere MQ, another server, and so on. In this case, look for messages indicating contention for resources; for example, on the z/OS console, you might see messages about DB2 deadlocks. In either case, the request times out waiting in the WLM queue to be dispatched in a servant (region).	The case where the threads are all busy processing requests could indicate one of the following conditions: • The number of servant regions that WLM may start is set too low. To set this value, on the administrative console select **Servers >Application Servers >*server_name* >Server Instance**. Click on **Multiple Instances Enabled** and specify a value for **Maximum Number of Instances**. • The number of threads allowed within a servant region is set too low. The number is controlled by the Isolation Policy setting in Administrative console or WebSphere variable: *server_region_ workload_profile* • You need to replicate servers to handle the amount of incoming work. All of these conditions indicate that performance tuning might be necessary.
Transaction	Possible causes of transaction timeouts include: • The same as those for WLM dispatch timeouts, or • Delays that prevent the transaction coordinator from committing or rolling back a transaction within the allotted time.	See the possible solutions for WLM dispatch timeouts. In addition, you can look for messages indicating contention for resources that are involved in the transaction that timed out.
Output	Possible causes of output timeouts are the same as those for WLM dispatch (dispatch is for IIOP, output is for HTTP).	See the possible solutions for WLM dispatch timeouts. In addition, you can use the WebSphere variable *protocol_accept_ http_work _after_min_srs=1* to prevent the HTTP transport handler from dispatching requests until WLM starts a minimum number of servant regions.

Guidelines for altering timeout values

This file lists common timer variables and tools for monitoring these timeout conditions

Generally speaking, increasing the timeout values should be your last resort, or only a temporary action taken to prevent multiple timeout-abend dumps from causing system performance problems. If you increase timeout values without properly diagnosing the timeout condition, the only results you might see are less frequent abends and dumps for the same timeout condition, or slower system or application performance.

For information on how to set values for these timer variables, and how these variables map to internal variables, see Controlling behavior through timeout values

Common timer variables and tools for monitoring these timeout conditions

WebSphere variable and its relationship, if any, to other timers	How to monitor processing for this type of timeout condition:	If you want to adjust the value, consider the following:
WLM timeout For HTTP work and Scalable Messaging Support, the WLM timer is not set and only the ConnectionResponseTimeout is in effect (covering the entire dispatch window)	SMF provides data on WLM queue time	How long work takes to get to a servant depends on the number of servants that WLM starts, how many you let it start, how many service classes the work is spread across, how much work you're getting, and so on.
ConnectionIOTimeOut None.	This behavior is not easily monitored. Turning on a trace point would indicate whether a client failed because of this input timeout setting, but tracing has performance consequences.	• How long are you willing to allow a control region worker thread to be blocked while it is waiting for a message? • How big are incoming HTTP requests? The larger they are, the longer it might take to get the whole request through the network.
ConnectionResponseTimeout If the application component starts transactions, then the transaction timers also might be involved.	This behavior is not easily monitored, but the controller will terminate the servant (region) with abend EC3 for this timeout condition.	• How long are you willing to let a client hang waiting for a response? • How long are you willing to let a thread in a servant (region) be tied up working on a response before concluding that the request has taken too long? • If you have multiple application threads in the servant (region), all of them will be terminated when only one of them times out. This loss of work might make you want to allow these time outs to occur less frequently.
ConnectionKeepAliveTimeout None. All the other timers relate to work processing, whereas this one relates to what happens when there is no work.	None.	How much time passes between requests vs. how much does it cost to establish a new session. You would want to keep idle sessions around for a while to avoid the startup cost of a new session, but don't want to keep them forever as resource usage accumulation will eventually be a problem.
Request Timeout (ORB Service) None. This variable is a client-side timeout, and IIOP only.	None, other than to observe the timeouts occuring on the client side.	How long are you willing to let the client wait?
ORB listener keep alive ORB SSL listener keep alive None. These variables relate to session activity during idle periods and only for IIOP, so these timers do not interact with the ConnectionKeepAliveTimeout timer.	You should read TCP/IP APAR PQ18618 for information about the `SOCK_TCP_KEEPALIVE` values and their consequences.	Is it useful to have idle sessions timeout? They normally don't which can consume resources. However, detecting a timeout requires network traffic between TCP/IP stacks. Creating traffic on otherwise idle sessions may have network consequences you don't want.
Total Transaction Lifetime Timeout This variable can be overridden by applications up to the maximum indicated by the Maximum Transaction Timeout variable, which limits the amount of time an application can set for its transactions to complete. Output timers also might cause work to time out, but the transaction timers and output timers are not aware of each other.	The controller issues message BBOT0003W to indicate a timeout condition, and terminates the servant (region) with abend EC3 reason codes 04130002 or 04130005.	• How long are you willing to let a client hang waiting for a response? • How long are you willing to let a thread in a servant (region) be tied up working on a response before concluding that the request has taken too long? • If you have multiple application threads in the servant (region), all of them will be terminated when only one of them times out. This loss of work might make you want to allow these time-outs to occur less frequently.

WebSphere variable and its relationship, if any, to other timers	How to monitor processing for this type of timeout condition:	If you want to adjust the value, consider the following:
Maximum Transaction Timeout If set, this variable limits the amount of time an application can set for its transactions to complete. If the Maximum Transaction Timeout variable is not set, application transactions are controlled by the time limit set on the Total Transaction Lifetime Timeout variable.	None.	Same considerations as for `transaction_ defaultTimeout`
transaction_ recoveryTimeout None	None.	Locks are held while one controller (region) waits for other controllers that are required to resolve in-doubt transactions. How long can you afford to have these resources held?

Chapter 9. Overview and new features for monitoring

Presentations from IBM Education Assistant

- Performance overview
- Performance advisors and Tivoli Performance Viewer (TPV)
- Request metrics

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:

Monitoring performance with third-party tools

- Enterprise Web Application Management WebSphere Performance Management Business Partner Solution Finder

  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
- 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.

Garbage collection

- 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.
- Tuning Garbage Collection with the 1.4.2 JavaTM Virtual Machine

  Learn more about using garbage collection in a Solaris operating environment.

Contents of this section: Monitoring

How do I monitor?

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

Documentation	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+

Enable and customize PMI data collection

Performance Monitoring Infrastructure (PMI) needs to be enabled (by default PMI is enabled out-of-the-box) 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. If the predefined statistic sets does not meet your monitoring requirement, the **Custom** option can be used to selectively enable or disable individual statistics

Documentation Show me Tell me

Monitor system health by viewing PMI data

The Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server without leaving the administrative console.

Documentation Show me Tell me

Measure application flow

Request metrics is a tool that allows you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components. The information tracked might either be saved to log files for later retrieval and analysis, be sent to Application Response Measurement (ARM) agents, or both.

Documentation Show me Tell me

Monitoring end user response time

Monitoring end user response time is an external perspective of how the overall Web site performs from an end user view and identifies how long the response time is for an end user. From this perspective, it is important to understand the load and response time on your site. To monitor at this level, many industry monitoring tools, for example, Tivoli Monitoring for Transaction Performance, support you to inject and monitor synthetic transactions, helping you identify when your Web site experiences a problem.

Monitoring overall system health

Monitoring overall system health is of fundamental importance to understand the health of every system involved that 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 have a rippling effect and cause the `servlet is slow` problem. IBM and several other business partners leverage the WebSphere APIs to capture this kind of performance data and to incorporate this data into an overall 24-by-7 monitoring solution across multiple products. 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 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	
JVM Memory	Use the JVM memory 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 in the administrative console to view this data.

Why use Tivoli Performance Viewer?

The Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server without leaving the administrative console.

In Version 4.0, Tivoli Performance Viewer was originally named the Resource Analyzer.

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 servlet response time

In addition to providing a built in viewer for PMI, TPV also allows you to view data for other products or customer applications that implement custom PMI. For more information on custom PMI, refer to “Enabling PMI data collection” on page 1630.

By providing the ability to look at this data, administrators can determine which part of the application to focus on to improve performance and what configuration settings to change to improve performance. For example, in order to determine what part of the application to focus on, you can view the summary charts for servlets, enterprise beans and Enterprise JavaBeans (EJB) methods, and sort these tables to determine which of these resources has the highest response time. You can then focus on improving the code path for those application resources taking the longest response time.

Likewise, you can look at the Tivoli Performance Viewer to help manage configuration settings through 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 pools need to be increased or decreased in size by monitoring the percent (%) usage. After configuration settings are changed based on that data, you can look at these views to determine the effectiveness of the changes. To help with configuration settings, the Tivoli Performance Viewer also provides an Advisor. The Advisor looks at various data while your application is running and provides advice on WebSphere configuration settings to change for improved performance.

TPV topologies and performance impacts

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 is affected by the various user and logging settings. “Using the Tivoli Performance Viewer” on page 1677 describes the user and logging settings and their effect on performance.

When the Tivoli Performance Viewer is running in a Network Deployment environment, the data is collected at each of the nodes and stored in memory at the node agent. Data is then viewed from the deployment manager. This architecture enables the monitoring work to be distributed among the nodes. Similar to the single server environment, the various user and logging settings directly influence the extent to which performance is affected.

Performance Monitoring Infrastructure (PMI)

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 4 and Version 5, PMI counters are enabled, based on a monitoring or instrumentation level. The levels are None, Low, Medium, High and Max (N, L, M, H, X). 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 Medium level enables all the counters at level M plus all the Low (L) level counters as well.

JSR-077 defines a set of statistics for J2EE components as part of the StatisticProvider interface. The Performance Monitoring Infrastructure (PMI) provides statistics using a monitoring level. The JSR-077 statistics do not match directly with the PMI monitoring levels. So enabling all the JSR-077 statistics *out-of-the-box* requires PMI to be enabled at a *high* level, which is not acceptable from the performance standpoint. The requirement is a performance overhead of 2 to 3 percent when frequently monitored statistics are enabled. In WebSphere Application Server Version 5.0, this is measured using the standard set. This poses a new requirement on PMI which is fine-grained control. Fine-grained control gets rid of the PMI levels and allows statistics to be enabled individually.

WebSphere Application Server Version 5.0 provided a set of interfaces for retrieving PMI information from Mbeans. JSR-77 defines an identical set of interfaces. In WebSphere Version 6.0, the new interfaces are the default returned from PMI when the stats attribute is retrieved. Additionally, we provide a simple migration tool to assist you with this change. The tool:

- Flags and optionally changes incorrect imports.
- Flags and optionally changes fully-qualifed class names.
- Changes uses of MessageBeanStats to MessageDrivenBeanStats (the WebSphere interface name differs from the J2EE interface name).

To get the old statistics classes returned, you need to set a system property (for example, a –D option) to websphereV5Statistics=true on the client. If this property is set, the PMI interface returns the WebSphere Application Server statistics object instead of an object that implements the J2EE interface.

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

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

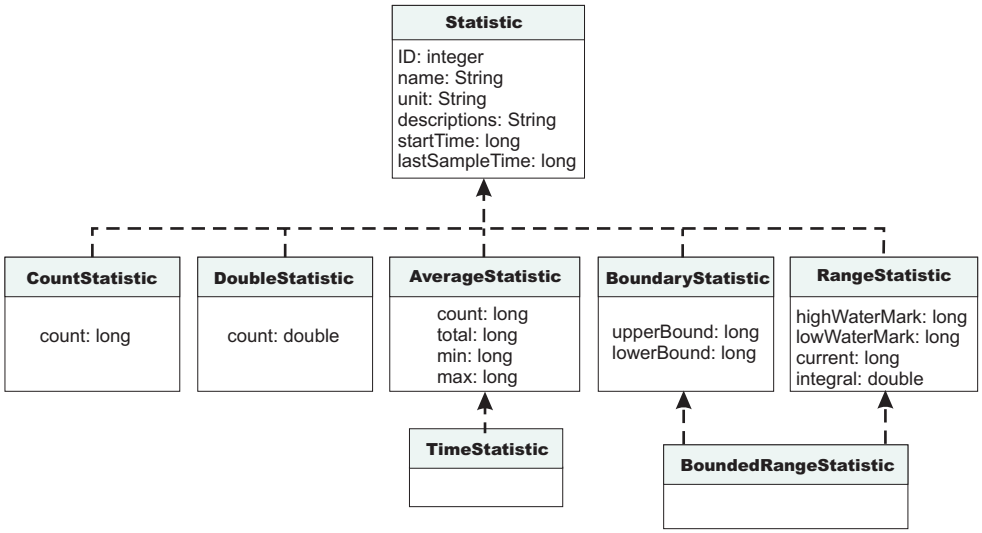
PMI provides server-side data collection and client-side API to retrieve performance data. Performance data has two components: static and dynamic.

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 Profiler Interface (JVMPI) 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.

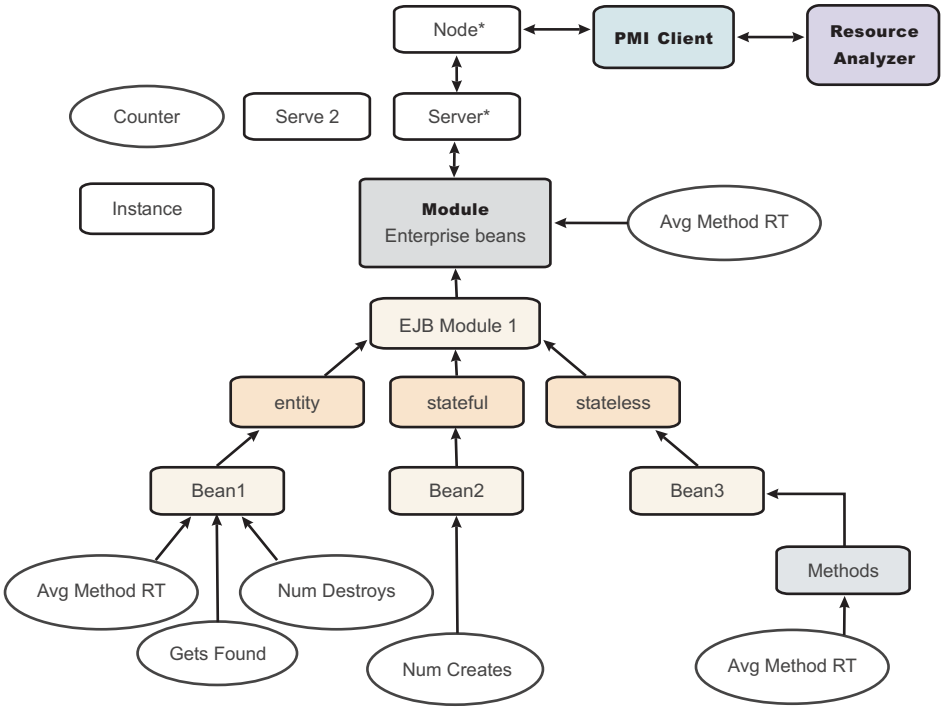
WebSphere Application Server, V6 deprecates the monitoring levels (*Low*, *Medium*, *High*, and *Max*) and introduces fine-grained control to enable/disable statistics individually. The fine-grained control is available under the custom option. Refer to “Enabling PMI using the administrative console” on page 1631 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 1631 for details.

PMI data organization

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 Servers. Each server organizes PMI data into modules and submodules.

Hierarchy of data collections used for performance reporting to Resource Analyzer



In the WebSphere Server Application, V6, Tivoli Performance Viewer is now a thin client integrated into the 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. For example, 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.

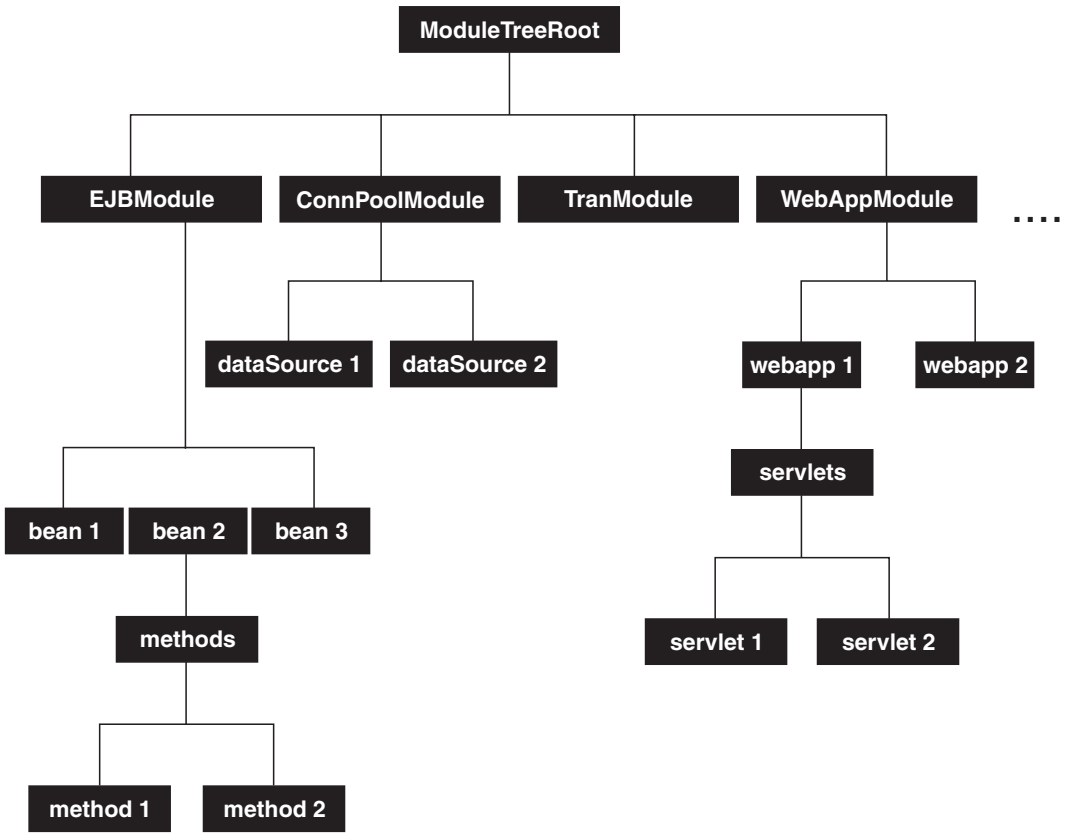
In WebSphere Application Server, V5.0, each counter has a specified monitoring level: none, low, medium, high or maximum. If the module is set to lower monitoring level than required by a particular counter, that counter is not enabled. Thus, if Bean1 has a medium monitoring level, Gets Found and Num Destroys are enabled because they require a low monitoring level. However, Avg Method RT is not enabled because it requires a high monitoring level.

In WebSphere Application Server, 5.0, the level Max is used to synchronize all the statistic updates. In WebSphere Application Server, V6.0, there are only two states for a statistic: enabled or disabled. In order to provide an option to enable synchronized updates, WebSphere Application Server, V6.0 provides a new configuration parameter, synchronizedUpdate, at the PMI service level. When this attribute is true all the statistic updates is synchronized, which is the equivalent to the Max level in V5.0. By default this is set to false.

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.

As part of fine-grained control feature, WebSphere V6.0 provides new 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 set to the *none* level. Choosing *basic* provides the J2EE and the top 34 statistics to give you a basic level of monitoring. Selecting *extended* gives you the basic level of monitoring plus WLM, Performance Advisor, and Tivoli resource models for a total of 61 statistics. Choosing *all* enables all statistics. Choosing *custom*, gives you fine-grained control to enable or disable statistics individually. **Note**: Do not change the module names.

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 modules.



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 connection pools for a database. Examples include the average size of the connection pool or number of connections, the average number of threads waiting for a connection, the average wait time in milliseconds for a connection, and the average time the connection is in use.

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.

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.

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.

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.

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 1639.

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
CreateCount	beanModule.creates	The number of times that beans were created	3.5.5 and later	Per home	CountStatistic	Low
RemoveCount	beanModule.removes	The number of times that beans were removed	3.5.5 and later	Per home	CountStatistic	Low
PassivateCount	beanModule.passivates	The number of times that beans were passivated (entity and stateful)	3.5.5 and later	Per home	CountStatistic	Low
ActivateCount	beanModule.activates	The number of times that beans were activated (entity and stateful)	3.5.5 and later	Per home	CountStatistic	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	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	Low
InstantiateCount	beanModule.instantiates	The number of times that bean objects were instantiated	3.5.5 and later	Per home	CountStatistic	Low
FreedCount	beanModule.destroys	The number of times that bean objects were freed	3.5.5 and later	Per home	CountStatistic	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	High
LiveCount	beanModule.concurrentLives	The number of concurrently live beans	3.5.5 and later	Per home	RangeStatistic	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	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	Max
LoadTime	beanModule.loadTime	The average time in milliseconds for loading the bean data from persistent storage (entity)	5.0	Per home	TimeStatistic	Medium
StoreTime	beanModule.storeTime	The average time in milliseconds for storing the bean data to persistent storage (entity)	5.0	Per home	TimeStatistic	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	Max
MethodCallCount	beanModule.totalMethodCalls	The total number of method calls	3.5.5 and later	Per home	CountStatistic	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	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	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	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	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	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	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	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	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	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	High
MessageCount	beanModule.messageCount	The number of messages delivered to the bean onMessage method (message driven beans)	5.0	Per type	CountStatistic	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	Low
WaitTime	beanModule.avgSrvSessionWaitTime	The average time to obtain a ServerSession from the pool (message-driven bean)	5.0	Per type	TimeStatistic	Medium
ServerSessionPoolUsage	beanModule.serverSessionUsage	The percentage of the server session pool in use (message driven)	5.0	Per type	RangeStatistic	High

Enterprise JavaBeans method counters:

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
MethodLevelCallCount	beanModule.methods.methodCalls	The number of calls to the bean methods (home, remote, local)	3.5.5 and later	Per method or per home	CountStatistic	Max
MethodLevelResponseTime	beanModule.methods.methodRt	The average response time in milliseconds on the bean methods (home, remote, local)	3.5.5 and later	Per method or per home	TimeStatistic	Max
MethodLevel ConcurrentInvocations	beanModule.method.methodLoad	The number of concurrent invocations to call a method	5.0	Per method or per home	RangeStatistic	Max

JDBC connection pool counters: Performance Monitoring Infrastructure (PMI) collects performance data for 4.0 and 5.0 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. You can find the data by using the Tivoli performance viewer and looking under each application server by click ***application_server*** > **JDBC connection pool**.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
CreateCount	connectionPoolModule.numCreates	The total number of connections created	3.5.5 and later	Per connection pool	CountStatistic	Low
PoolSize	connectionPoolModule.poolSize	The size of the connection pool	3.5.5 and later	Per connection pool	BoundedRangeStatistic	High
FreePoolSize	connectionPoolModule.freePoolSize	The number of free connections in the pool	5.0	Per connection pool	BoundedRangeStatistic	High
AllocateCount	connectionPoolModule.numAllocates	The total number of connections allocated	3.5.5 and later	Per connection pool	CountStatistic	Low
ReturnCount	connectionPoolModule.numReturns	The total number of connections returned	4.0 and later	Per connection pool	CountStatistic	Low
WaitingThread Count	connectionPoolModule.concurrentWaiters	The number of threads that are currently waiting for a connection	3.5.5 and later	Per connection pool	RangeStatistic	High
FaultCount	connectionPoolModule.faults	The total number of faults, such as timeouts, in the connection pool	3.5.5 and later	Per connection pool	CountStatistic	Low
CloseCount	connectionPoolModule.numDestroys	The total number of connections closed.	3.5.5 and later	Per connection pool	CountStatistic	Low
WaitTime	connectionPoolModule.avgWaitTime	The average waiting time in milliseconds until a connection is granted	5.0	Per connection pool	TimeStatistic	Medium
UseTime	connectionPoolModule.avgUseTime	The average time a connection is used (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	Medium
PercentUsed	connectionPoolModule.percentUsed	The average percent of the pool that is in use	3.5.5 and later	Per connection pool	RangeStatistic	High
PercentMaxed	connectionPoolModule.percentMaxed	The average percent of the time that all connections are in use	3.5.5 and later	Per connection pool	RangeStatistic	High
PrepStmtCache DiscardCount	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	Low

Managed Connection Count	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	Low
Connection HandleCount	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	Low
JDBCTime	connectionPoolModule.jdbcOperationTimer	The amount of time in milliseconds spent running in the JDBC driver (includes time spent in the JDBC driver, network, and database)	5.0	Per data source	TimeStatistic	Medium

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
Managed Connection Count	j2cModule.numManagedConnections	The number of ManagedConnection objects in use	5.0	Per connection factory	CountStatistic	Low
Connection Handle Count	j2cModule.numConnectionHandles	The number of connections that are associated with ManagedConnections (physical connections) objects in this pool	5.0	Per connection factory	CountStatistic	Low
CreateCount	j2cModule.numManagedConnectionsCreated	The total number of managed connections created	5.0	Per connection factory	CountStatistic	Low
CloseCount	j2cModule.numManagedConnectionsDestroyed	The total number of managed connections destroyed	5.0	Per connection factory	CountStatistic	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	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	Low
FaultCount	j2cModule.faults	The number of faults, such as timeouts, in the connection pool	5.0	Per connection factory	CountStatistic	Low
FreePoolSize	j2cModule.freePoolSize	The number of free connections in the pool	5.0	Per connection factory	BoundedRangeStatistic	High

PoolSize	j2cModule.poolSize	Average number of managed connections in the pool.	5.0	Per connection factory	BoundedRangeStatistic	High
WaitingThreadCount	j2cModule.concurrentWaiters	Average number of threads concurrently waiting for a connection	5.0	Per connection factory	RangeStatistic	High
PercentUsed	j2cModule.percentUsed	Average percent of the pool that is in use	5.0	Per connection factory	RangeStatistic	High
PercentMaxed	j2cModule.percentMaxed	Average percent of the time that all connections are in use	5.0	Per connection factory	RangeStatistic	High
WaitTime	j2cModule.avgWait	Average waiting time in milliseconds until a connection is granted	5.0	Per connection factory	TimeStatistic	Medium
UseTime	j2cModule.useTime	Average time in milliseconds that connections are in use	5.0	Per connection factory	TimeStatistic	Medium

Servlet session counters: Data counters for this category contain usage information for HTTP sessions.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
CreateCount	servletSessionsModule.createdSessions	The number of sessions that were created	3.5.5 and later	Per Web application	CountStatistic	Low
InvalidateCount	servletSessionsModule.invalidatedSessions	The number of sessions that were invalidated	3.5.5 and later	Per Web application	CountStatistic	Low
LifeTime	servletSessionsModule.sessionLifeTime	The average session life time in milliseconds (time invalidated - time created)	3.5.5 and later	Per Web application	TimeStatistic	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	High
LiveCount	servletSessionsModule.liveSessions	The number of sessions that are currently cached in memory	5.0 and later	Per Web application	RangeStatistic	High
NoRoomFor NewSession Count	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	Low
CacheDiscard Count	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	Low

ExternalRead Time	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	Medium
ExternalRead Size	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	Medium
ExternalWrite Time	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	Medium
ExternalWrite Size	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	Medium
AffinityBreak Count	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	Low
SessionObject Size	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	Max
TimeSinceLast Activated	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	Medium
Timeout Invalidation Count	servletSessionsModule.invalidatedViaTimeout	The number of sessions that are invalidated by timeout.	5.0	Per Web application	CountStatistic	Low

ActivateNon ExistSession Count	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	Low

Transaction counters:

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
GlobalBegunCount	transactionModule.globalTransBegun	The total number of global transactions started on the server	4.0 and later	Per transaction manager or server	CountStatistic	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	Low
LocalBegunCount	transactionModule.localTransBegun	The total number of local transactions started on the server	4.0 and later	Per transaction manager or server	CountStatistic	Low
ActiveCount	transactionModule.activeGlobalTrans	The number of concurrently active global transactions	3.5.5 and later	Per transaction manager or server	CountStatistic	Low
LocalActiveCount	transactionModule.activeLocalTrans	The number of concurrently active local transactions	4.0 and later	Per transaction manager or server	CountStatistic	Low
GlobalTranTime	transactionModule.globalTranDuration	The average duration of global transactions	3.5.5 and later	Per transaction manager or server	TimeStatistic	Medium
LocalTranTime	transactionModule.localTranDuration	The average duration of local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
GlobalBefore CompletionTime	transactionModule.globalBefore CompletionDuration	The average duration of before_completion for global transactions	4.0 and later	per transaction manager or server	TimeStatistic	Medium
GlobalCommitTime	transactionModule.globalCommitDuration	The average duration of commit for global transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
GlobalPrepareTime	transactionModule.globalPrepareDuration	The average duration of prepare for global transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
LocalBefore CompletionTime	transactionModule.localBefore CompletionDuration	The average duration of before_completion for local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
LocalCommitTime	transactionModule.localCommitDuration	The average duration of commit for local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium

CommittedCount	transactionModule.globalTransCommitted	The total number of global transactions committed	3.5.5 and later	Per transaction manager or server	CountStatistic	Low
RolledbackCount	transactionModule.globalTransRolledBack	The total number of global transactions rolled back	3.5.5 and later	Per transaction manager or server	CountStatistic	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	Low
LocalCommittedCount	transactionModule.localTransCommitted	The number of local transactions committed	4.0 and later	Per transaction manager or server	CountStatistic	Low
LocalRolledbackCount	transactionModule.localTransRolledBack	The number of local transactions rolled back	4.0 and later	Per transaction manager or server	CountStatistic	Low
GlobalTimeoutCount	transactionModule.globalTransTimeout	The number of global transactions timed out	4.0 and later	Per transaction manager or server	CountStatistic	Low
LocalTimeoutCount	transactionModule.localTransTimeout	The number of local transactions timed out	4.0 and later	Per transaction manager or server	CountStatistic	Low

Web application counters: Data counters for this category contain information for the selected server.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
LoadedServletCount	webAppModule.numLoadedServlets	The number of loaded servlets	3.5.5 and later	Per Web application	CountStatistic	Low
ReloadCount	webAppModule.numReloads	The number of reloaded servlets	3.5.5 and later	Per Web application	CountStatistic	Low
RequestCount	webAppModule.servlets.totalRequests	Total number of requests that a servlet processed	3.5.5 and later	Per servlet	CountStatistic	Low
Concurrent Requests	webAppModule.servlets.concurrentRequests	The number of requests that are concurrently processed	3.5.5 and later	Per servlet	RangeStatistic	High
ServiceTime	webAppModule.servlets.responseTime	The response time, in milliseconds, of a servlet request	3.5.5 and later	Per servlet	TimeStatistic	Medium
ErrorCount	webAppModule.servlets.numErrors	Total number of errors in a servlet or JavaServer Page (JSP)	3.5.5 and later	Per servlet	CountStatistic	Low

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 1379.

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
MaxInMemoryCache EntryCount	cacheModule.maxInMemoryCacheEntryCount	The maximum number of in-memory cache entries.	5.0 and above	Per server	CountStatistic	Low
InMemoryCache EntryCount	cacheModule.inMemoryCacheEntryCount	The current number of in-memory cache entries.	5.0 and above	Per server	CountStatistic	Low

Template counters:

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
HitsInMemoryCount	cacheModule.hitsInMemoryCount	The count of requests for cacheable objects that are served from memory	5.0 and above	per cache instance	CountStatistic	Low
HitsOnDiskCount	cacheModule.hitsOnDiskCount	The count of requests for cacheable objects that are served from disk	5.0 and above	per cache instance	CountStatistic	Low
ExplicitInvalidationCount	cacheModule.explicitInvalidationCount	The count of explicit invalidations	5.0 and above	per cache instance	CountStatistic	Low
LruInvalidationCount	cacheModule.lruInvalidationCount	The count of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm	5.0 and above	per cache instance	CountStatistic	Low
TimeoutInvalidationCount	cacheModule.timeoutInvalidationCount	The count of cache entries that are removed from memory and disk because their timeout has expired	5.0 and above	per cache instance	CountStatistic	Low
InMemoryAndDisk CacheEntryCount	cacheModule.inMemoryAnd DiskCacheEntryCount	The current number of used cache entries in memory and disk	5.0 and above	per cache instance	CountStatistic	Low
RemoteHitCount	cacheModule.remoteHitCount	The count of requests for cacheable objects that are served from other Java virtual machines within the replication domain	5.0 and above	per cache instance	CountStatistic	Low
MissCount	cacheModule.missCount	The count of requests for cacheable objects that were not found in the cache	5.0 and above	per cache instance	CountStatistic	Low
ClientRequestCount	cacheModule.clientRequestCount	The count of requests for cacheable objects that are generated by applications running on this application server	5.0 and above	per cache instance	CountStatistic	Low

DistributedRequestCount	cacheModule.distributedRequestCount	The count of requests for cacheable objects that are generated by cooperating caches in this replication domain	5.0 and above	per cache instance	CountStatistic	Low
ExplicitMemoryInvalidationCount)	cacheModule.explicitMemoryInvalidationCount	The count of explicit invalidations resulting in the removal of an entry from memory	5.0 and above	per cache instance	CountStatistic	Low
ExplicitDiskInvalidationCount	cacheModule.explicitDiskInvalidationCount	The count of explicit invalidations resulting in the removal of an entry from disk	5.0 and above	per cache instance	CountStatistic	Low
LocalExplicitInvalidationCount	cacheModule.localExplicitInvalidationCount	The count of explicit invalidations generated locally, either programmatically or by a cache policy	5.0 and above	per cache instance	CountStatistic	Low
RemoteExplicitInvalidationCount	cacheModule.remoteExplicitInvalidationCount	The count of explicit invalidations received from a cooperating Java virtual machine in this replication domain	5.0 and above	per cache instance	CountStatistic	Low
RemoteCreationCount	cacheModule.remoteCreationCount	The number of cache entries that are received from cooperating dynamic caches	5.0 and above	per cache instance	CountStatistic	Low

Web services counters: Counters for this category contain information for the Web services.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
LoadedWebService Count	webServicesModule.numLoadedServices	The number of loaded Web services	5.02 and above	Per service	CountStatistic	Low
ReceivedRequest Count	webServicesModule.services.numberReceived	The number of requests the service received	5.02 and above	Per Web service	CountStatistic	Low
DispatchedRequest Count	webServicesModule.services.numberDispatched	The number of requests the service dispatched or delivered	5.02 and above	Per Web service	CountStatistic	Low
ProcessedRequestCount	webServicesModule.services.numberSuccessfull	The number of requests the service successfully processed	5.02 and above	Per Web service	TimeStatistic	Low
ResponseTime	webServicesModule.services.responseTime	The average response time, in milliseconds, for a successful request	5.02 and above	Per Web service	TimeStatistic	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	Medium
DispatchResponseTime	webServicesModule.services.dispatchResponse Time	The average response time, in milliseconds, to dispatch a request	5.02 and above	Per Web service	TimeStatistic	Medium

ReplyResponseTime	webServicesModule.services.replyResponse Time	The average response time, in milliseconds, to prepare a reply after dispatch	5.02 and above	Per Web service	TimeStatistic	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	Medium
RequestPayloadSize	webServicesModule.services.requestSize	The average payload size in bytes of a request	5.02 and above	Per Web service	TimeStatistic	Medium
ReplyPayloadSize	webServicesModule.services.replySize	The average payload size in bytes of a reply	5.02 and above	Per Web service	TimeStatistic	Medium

High availability manager counters:

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
Number of local groups	hamanagermodule.numLocalGroups	The total number of local groups.	6.0 and above	Per server	RangeStatistic	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	High
Number of bulletin-board subjects	hamanagermodule.bbMgrNumSubjects	The total number of subjects managed.	6.0 and above	Per server	RangeStatistic	High
Number of bulletin-board subscriptions	hamanagermodule.bbMgrNumSubscriptions	The total number of bulletin-board subscriptions.	6.0 and above	Per server	RangeStatistic	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	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	High
Number of local bulletin-board subscriptions	hamanagermodule.bbLocalNumSubscriptions	The total number of local subject subscriptions. The number includes the proxy subscriptions (if any) done by the core group bridge service on behalf of servers belonging to different WebSphere cells.	6.0 and above	Per server	TimeStatistic	High

DCS stack counters:

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level
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	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	High
Number of sent messages	DCSStats.outgoingMessageCounter	Number of messages sent through the DCS stack	6.0 and above	Per DCS stack	CountStatistic	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	High
Number of received messages	DCSStats.incomingMessageCounter	Number of messages received by the DCS stack	6.0 and above	Per DCS stack	CountStatistic	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	High
Number of messages retransmitted by local member during the view change	DCSStats.numOfVSCompletionMessages	Number of messages that were retransmitted during the view change to ensure synchronization with other members.	6.0 and above	Per DCS stack	AverageStatistic	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	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	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	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	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	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	High
Number of view changes	DCSStats.viewCounter	Number of times that this member underwent view changes.	6.0 and above	Per DCS stack	CountStatistic	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	Medium

System Integration Bus (SIB) and Messaging counters: For information on SIB counters, see the following articles:

- “MessageStore Statistics”
- “Mediation Framework Statistics” on page 1604
- “Message Processor Statistics” on page 1604
- “Communications statistics” on page 1610

MessageStore Statistics:

Counter definitions*:* Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache

Name	Key	Description	Version	Granularity	Type	Level
CacheAddStored Count	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	Per Messaging Engine	CountStatistic	High

CacheAddNot StoredCount	MessageStoreStats.CacheAddNotStoredCount	The number of items that have been added to the message store during the current session that are not persistent	6.0	Per Messaging Engine	CountStatistic	High
CacheUpdate StoredCount	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	Per Messaging Engine	CountStatistic	High
CacheUpdate NotStoredCount	MessageStoreStats.CacheUpdateNotStoredCount	The number of items that have been updated in the message store during the current session that not persistent	6.0	Per Messaging Engine	CountStatistic	High
CacheRemove StoredCount	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	Per Messaging Engine	CountStatistic	High
CacheRemove NotStoredCount	MessageStoreStats.CacheRemoveNotStoredCount	The number of items that have been removed from the message store during the current session that are not persistent	6.0	Per Messaging Engine	CountStatistic	High
CacheRestore Count	MessageStoreStats.CacheRestoreCount	The number of items restored to memory from persistence during the current session	6.0	Per Messaging Engine	CountStatistic	High
CacheCurrent StoredCount	MessageStoreStats.CacheCurrentStoredCount	The number of items currently in the dynamic memory cache which are either persistent or potentially persistent	6.0	Per Messaging Engine	CountStatistic	High
CacheCurrent NotStoredCount	MessageStoreStats.CacheCurrentNotStoredCount	The number of items currently in the dynamic memory cache which are never persisted	6.0	Per Messaging Engine	CountStatistic	High
CacheCurrent StoredByteCount	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	Per Messaging Engine	CountStatistic	High

CacheCurrent NotStoredByte Count	MessageStoreStats.CacheCurrentNotStoredByteCount	The current total of the declared sizes of all items in the dynamic memory cache which are never persisted	6.0	Per Messaging Engine	CountStatistic	High
CacheTotal StoredCount	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	Per Messaging Engine	CountStatistic	High
CacheTotalNot StoredCount	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	Per Messaging Engine	CountStatistic	High
CacheTotalStored ByteCount	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	Per Messaging Engine	CountStatistic	High
CacheTotalNot StoredByteCount	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	Per Messaging Engine	CountStatistic	High
CacheStored DiscardCount	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	Per Messaging Engine	CountStatistic	High
CacheNotStored DiscardCount	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	Per Messaging Engine	CountStatistic	High

CacheStored DiscardByteCount	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	Per Messaging Engine	CountStatistic	High
CacheNotStored DiscardByteCount	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	Per Messaging Engine	CountStatistic	High
CacheStored RefusalCount	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	Per Messaging Engine	CountStatistic	High
CacheNotStored RefusalCount	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	Per Messaging Engine	CountStatistic	High
CacheStream SpillingCount	MessageStoreStats.CacheStreamSpillingCount	Number of streams currently spilling potentially persistent items	6.0	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Cache

Name	Key	Description	Version	Granularity	Type	Level
SpillDispatcher RequestSize	MessageStoreStats.SpillDispatcherRequestSize	Measures the number of operations on nonpersistent data dispatched for spilling to the data store.	6.0	Per Messaging Engine	AverageStatistic	High
SpillDispatcher BatchSize	MessageStoreStats.SpillDispatcherBatchSize	Measures the batching of operations on nonpersistent data dispatched for spilling to the data store.	6.0	Per Messaging Engine	AverageStatistic	High

SpillDispatcher AvoidanceCount	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	Per Messaging Engine	AverageStatistic	High
SpillDispatcher AvoidanceSize	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	Per Messaging Engine	AverageStatistic	High
PersistentDispatcher RequestSize	MessageStoreStats.PersistentDispatcherRequestSize	Measures the number of operations on reliable persistent data dispatched for writing to the data store.	6.0	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	Per Messaging Engine	AverageStatistic	High
PersistentDispatcher CancellationCount	MessageStoreStats.PersistentDispatcher CancellationCount	Counts the number of global transaction completion phases whose operations cancelled out before being written to the data store.	6.0	Per Messaging Engine	AverageStatistic	High
PersistentDispatcher AvoidanceCount	MessageStoreStats.PersistentDispatcher AvoidanceCount	Measures the number of operations on reliable persistent data dispatched for writing to the data store but whose writing was subsequently unnecessary.	6.0	Per Messaging Engine	AverageStatistic	High
PersistentDispatcher AvoidanceSize	MessageStoreStats.PersistentDispatcher AvoidanceSize	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	Per Messaging Engine	AverageStatistic	High
JDBCOpenCount	MessageStoreStats.JDBCOpenCount	JDBC connections open	6.0	Per Messaging Engine	CountStatistic	High
JDBCTransaction CompleteCount	MessageStoreStats.JDBCTransactionCompleteCount	JDBC local transactions completed	6.0	Per Messaging Engine	CountStatistic	High

JDBCTransaction CompleteCount	MessageStoreStats.JDBCTransactionCompleteCount	JDBC local transactions completed	6.0	Per Messaging Engine	CountStatistic	High
JDBCTransaction AbortCount	MessageStoreStats.JDBCTransactionAbortCount	JDBC local transactions aborted	6.0	Per Messaging Engine	CountStatistic	High
JDBCTransaction Time	MessageStoreStats.JDBCTransactionTime	Total execution time of internal batches	6.0	Per Messaging Engine	TimeStatistic	High
JDBCItemInsert Count	MessageStoreStats.JDBCItemInsertCount	JDBC Item table inserts	6.0	Per Messaging Engine	CountStatistic	High
JDBCItemDelete Count	MessageStoreStats.JDBCItemDeleteCount	JDBC Item table deletes	6.0	Per Messaging Engine	CountStatistic	High
JDBCItemUpdate Count	MessageStoreStats.JDBCItemUpdateCount	JDBC Item table updates	6.0	Per Messaging Engine	CountStatistic	High
ItemInsertBatch Count	MessageStoreStats.ItemInsertBatchCount	Item table insert batches	6.0	Per Messaging Engine	CountStatistic	High
ItemDeleteBatch Count	MessageStoreStats.ItemDeleteBatchCount	Item table delete batches	6.0	Per Messaging Engine	CountStatistic	High
ItemUpdateBatch Count	MessageStoreStats.ItemUpdateBatchCount	Item table update batches	6.0	Per Messaging Engine	CountStatistic	High
JDBCTransaction InsertCount	MessageStoreStats.JDBCTransactionInsertCount	JDBC transaction table inserts	6.0	Per Messaging Engine	CountStatistic	High
JDBCTransaction DeleteCount	MessageStoreStats.JDBCTransactionDeleteCount	JDBC transaction table deletes	6.0	Per Messaging Engine	CountStatistic	High
JDBCTransaction UpdateCount	MessageStoreStats.JDBCTransactionUpdateCount	JDBC transaction table updates	6.0	Per Messaging Engine	CountStatistic	High
TransactionInsert BatchCount	MessageStoreStats.TransactionInsertBatchCount	Transaction table insert batches	6.0	Per Messaging Engine	CountStatistic	High
TransactionDelete BatchCount	MessageStoreStats.TransactionDeleteBatchCount	Transaction table delete batches	6.0	Per Messaging Engine	CountStatistic	High
TransactionUpdate BatchCount	MessageStoreStats.TransactionUpdateBatchCount	Transaction table update batches	6.0	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Expiry

Name	Key	Description	Version	Granularity	Type	Level
ExpiryIndex ItemCount	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	Per Messaging Engine	CountStatistic	High

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Storage Management > Transactions

Name	Key	Description	Version	Granularity	Type	Level

LocalTransaction StartCount	MessageStoreStats.LocalTransactionStartCount	Local transactions started	6.0	Per Messaging Engine	CountStatistic	High
LocalTransaction AbortCount	MessageStoreStats.LocalTransactionAbortCount	Local transactions aborted	6.0	Per Messaging Engine	CountStatistic	High
LocalTransaction CommitCount	MessageStoreStats.LocalTransactionCommitCount	Local transactions committed	6.0	Per Messaging Engine	CountStatistic	High
GlobalTransaction StartCount	MessageStoreStats.GlobalTransactionStartCount	Global transactions started	6.0	Per Messaging Engine	CountStatistic	High
GlobalTransaction InDoubtCount	MessageStoreStats.GlobalTransactionInDoubtCount	Global transactions in doubt	6.0	Per Messaging Engine	CountStatistic	High
GlobalTransaction AbortCount	MessageStoreStats.GlobalTransactionAbortCount	Global transactions aborted	6.0	Per Messaging Engine	CountStatistic	High
GlobalTransaction CommitCount	MessageStoreStats.GlobalTransactionCommitCount	Global transactions committed	6.0	Per Messaging Engine	CountStatistic	High

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	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	per mediated destination	TimeStatistic	Low
MediatedMessagesCount	Mediation.MediatedMessageCount	The number of messages that have been mediated at a mediated destination.	6.0	per mediated destination	CountStatistic	Low

Message Processor Statistics:

Counter definitions*:* Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Queues

Name	Key	Description	Version	Granularity	Type	Level

Available Message Count	QueueStats.Available MessageCount	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	Per destination	CountStatistic	Low
Unavailable Message Count	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	Per destination	CountStatistic	Low
Local Producer Attaches	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	Per destination	CountStatistic	Low
Local Producer Count	QueueStats.LocalProducerCount	The number of currently attached local producers.	6.0	Per destination	CountStatistic	Low
Local Consumer Attaches	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	Per destination	CountStatistic	Low
Local Consumer Count	QueueStats.LocalConsumerCount	The number of currently attached local consumers.	6.0	Per destination	CountStatistic	Low
Total Messages Produced	QueueStats.TotalMessagesProducedCount	The total number of messages produced to this queue, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Best Effort Non-persistent Messages Produced	QueueStats.BestEffortNonPersistentMessagesProduced Count	The number of Best Effort Non-persistent messages produced, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Express Non-persistent Messages Produced	QueueStats.ExpressNonPersistentMessagesProducedCount	The number of Express Non-persistent messages produced, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Reliable Non-persistent Messages Produced	QueueStats.ReliableNonPersistentMessagesProducedCount	The number of Reliable Non-persistent messages produced, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Reliable Persistent Messages Produced	QueueStats.ReliablePersistentMessagesProducedCount	The number of Reliable Persistent messages produced, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low

Assured Persistent Messages Produced	QueueStats.AssuredPersistentMessagesProducedCount	The number of Assured Persistent messages produced, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Total Messages Consumed	QueueStats.TotalMessagesConsumedCount	The total number of messages consumed from this queue, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Best Effort Non-persistent Messages Consumed	QueueStats.BestEffortNonPersistentMessagesConsumedCount	The number of Best Effort Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Express Non-persistent Messages Consumed	QueueStats.ExpressNonPersistentMessagesConsumedCount	The number of Express Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Reliable Non-persistent Messages Consumed	QueueStats.ReliableNonPersistentMessagesConsumedCount	The number of Reliable Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Reliable Persistent Messages Consumed	QueueStats.ReliablePersistentMessagesConsumedCount	The number of Reliable Persistent messages consumed, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Assured Persistent Messages Consumed	QueueStats.AssuredPersistentMessagesConsumedCount	The number of Assured Persistent messages consumed, for the lifetime of this messaging engine.	6.0	Per destination	CountStatistic	Low
Report Enabled Messages Expired.	QueueStats.ReportEnabledMessagesExpiredCount	The number of report enabled messages that expired while on this queue.	6.0	Per destination	CountStatistic	Low
Aggregate Message Wait Time	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	Per destination	CountStatistic	Low
Local Message Wait Time	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	Per destination	CountStatistic	Low

Local Oldest Message Age	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	Per destination	CountStatistic	Low

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Topicspaces

Name	Key	Description	Version	Granularity	Type	Level
Incomplete Publication Count	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	Per destination	CountStatistic	Low
Local Publisher Attaches	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	Per destination	CountStatistic	Low
Local Publisher Count	TopicspaceStats.LocalPublisherCount	The number of local publishers to topics in this topicspace.	6.0	Per destination	CountStatistic	Low
Total Local Subscription Count	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	Per destination	CountStatistic	Low
Non-Durable Local Subscription Count	TopicspaceStats.NonDurableLocalSubscriptionCount	The number of non-durable subscriptions.	6.0	Per destination	CountStatistic	Low
Durable Local Subscription Count	TopicspaceStats.DurableLocalSubscriptionCount	The number of durable subscriptions.	6.0	Per destination	CountStatistic	Low
Total Messages Published	TopicspaceStats.TotalMessagesPublished Count	The total number of publications to this topicspace.	6.0	Per destination	CountStatistic	Low
BestEffort Non-persistent Messages Published	TopicspaceStats.BestEffortNonPersistentMessages PublishedCount	The number of Best Effort Non-persistent messages published	6.0	Per destination	CountStatistic	Low
Express Non-persistent Messages Published	TopicspaceStats.ExpressNonPersistentMessagesPublished Count	The number of Express Non-persistent messages published	6.0	Per destination	CountStatistic	Low
Reliable Non-persistent Messages Published	TopicspaceStats.ReliableNonPersistentMessagesPublished Count	The number of Reliable Non-persistent messages published	6.0	Per destination	CountStatistic	Low

Reliable Persistent Messages Published	TopicspaceStats.ReliablePersistentMessagesPublishedCount	The number of Reliable Persistent messages published	6.0	Per destination	CountStatistic	Low
Assured Persistent Messages Published	TopicspaceStats.AssuredPersistentMessagesPublishedCount	The number of Assured Persistent messages published	6.0	Per destination	CountStatistic	Low
Total Local Subscription Hits	TopicspaceStats.TotalLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched topicspace publications.	6.0	Per destination	CountStatistic	Low
Best Effort Non-persistent Local Subscription Hits	TopicspaceStats.BestEffortNonPersistentLocalSubscription HitCount	The cumulative total of subscriptions which have matched Best Effort Non-persistent publications.	6.0	Per destination	CountStatistic	Low
Express Non-persistent Local Subscription Hits	TopicspaceStats.ExpressNonPersistentLocalSubscription HitCount	The cumulative total of subscriptions which have matched Express Non-persistent publications.	6.0	Per destination	CountStatistic	Low
Reliable Non-persistent Local Subscription Hits	TopicspaceStats.ReliableNonPersistentLocalSubscription HitCount	The cumulative total of subscriptions which have matched Reliable Non-persistent publications.	6.0	Per destination	CountStatistic	Low
Reliable Persistent Local Subscription Hits	TopicspaceStats.ReliablePersistentLocalSubscription HitCount	The cumulative total of subscriptions which have matched Reliable Persistent publications.	6.0	Per destination	CountStatistic	Low
Assured Persistent Local Subscription Hits	TopicspaceStats.AssuredPersistentLocalSubscriptionHitCount	The cumulative total of subscriptions which have matched Assured Persistent publications.	6.0	Per destination	CountStatistic	Low
Report Enabled Publications Expired	TopicspaceStats.ReportEnabledPublicationsExpiredCount	The number of report enabled incomplete publications that expired while on this topicspace.	6.0	Per destination	CountStatistic	Low
Local Oldest Publication	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	Per destination	TimeStatistic	max

Performance Modules > SIB Service > SIB Messaging Engines > Messaging Engine > Destinations > Topicspaces > Topic > DurableSubscriptions

Name	Key	Description	Version	Granularity	Type	Level

Available Message Count	DurableSubscriptionStats.AvailableMessageCount	The number of messages waiting to be consumed.	6.0	per mediated destination	CountStatistic	Low
Total Messages Consumed	DurableSubscriptionStats.TotalMessages ConsumedCount	The total number of messages consumed from this durable subscription.	6.0	per mediated destination	CountStatistic	Low
Best Effort Non-persistent Messages Consumed	DurableSubscriptionStats.BestEffortNonPersistentMessages ConsumedCount	The number of Best Effort Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0	per mediated destination	CountStatistic	Low
Express Non-persistent Messages Consumed	DurableSubscriptionStats.ExpressNonPersistentMessages ConsumedCount	The number of Express Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0	per mediated destination	CountStatistic	Low
Reliable Non-persistent Messages Consumed	DurableSubscriptionStats.ReliableNonPersistentMessages ConsumedCount	The number of Reliable Non-persistent messages consumed, for the lifetime of this messaging engine.	6.0	per mediated destination	CountStatistic	Low
Reliable Persistent Messages Consumed	DurableSubscriptionStats.ReliablePersistentMessages ConsumedCount	The number of Reliable Persistent messages consumed, for the lifetime of this messaging engine.	6.0	per mediated destination	CountStatistic	Low
Assured Persistent Messages Consumed	DurableSubscriptionStats.AssuredPersistentMessages ConsumedCount	The number of Assured Persistent messages consumed, for the lifetime of this messaging engine.	6.0	per mediated destination	CountStatistic	Low
Aggregate Message Wait Time	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	per mediated destination	TimeStatistic	High
Local Message Wait Time	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	per mediated destination	TimeStatistic	High
Local Oldest Publication	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	per mediated destination	TimeStatistic	Max

			6.0	per mediated destination	CountStatistic	Low
			6.0	per mediated destination	CountStatistic	Low
			6.0	per mediated destination	CountStatistic	Low
			6.0	per mediated destination	CountStatistic	Low
			6.0	per mediated destination	CountStatistic	Low

Communications statistics:

Counter definitions*:* Performance Modules > SIB Service > SIB Communications > Clients > Detailed Statistics

Name	Key	Description	Version	Granularity	Type	Level
BytesSentAtHighest PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtVeryHigh PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtHigh PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low

BytesSentAtJMS9 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS8 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS7 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low

BytesSentAtJMS6 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS5 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS4 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low

BytesSentAtJMS3 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS2 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS1 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low

BytesSentAtJMS0 PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtLow PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtVeryLow PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtLowest PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low

BytesReceivedAtHighest PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtVery HighPriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtHigh PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS9 PriorityCount	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	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	All clients connected or that have been connected to this application server	CountStatistic	Low
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	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	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	All clients connected or that have been connected to this application server	CountStatistic	Low
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	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	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	All clients connected or that have been connected to this application server	CountStatistic	Low
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	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	All clients connected or that have been connected to this application server	CountStatistic	Low

BytesReceivedAtLow PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtVery LowPriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtLowest PriorityCount	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	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS9 PriorityCount	ClientDetailedStats.MessagesSentAtJMS9Priority	Number of messages transmitted at JMS priority 9.	6.0	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS8 PriorityCount	ClientDetailedStats.MessagesSentAtJMS8Priority	Number of messages transmitted at JMS priority 8.	6.0	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS7 PriorityCount	ClientDetailedStats.MessagesSentAtJMS7Priority	Number of messages transmitted at JMS priority 7.	6.0	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS6 PriorityCount	ClientDetailedStats.MessagesSentAtJMS6Priority	Number of messages transmitted at JMS priority 6.	6.0	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	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	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	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS2PriorityCount	ClientDetailedStats.MessagesSentAtJMS2Priority	Number of messages transmitted at JMS priority 2.	6.0	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	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	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	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	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	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	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	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	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS3PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS3Priority	Number of messages received at JMS priority 3.	6.0	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	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	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	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	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	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	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	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	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	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	All clients connected or that have been connected to this application server	CountStatistic	Low
MulticastWrite BytesCount	ClientStats.MulticastWriteBytes	Number of bytes transmitted using multicast protocols.	6.0	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BufferedWriteBytes Count	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	All clients connected or that have been connected to this application server	CountStatistic	Low
BufferedReadBytes Count	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	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesBytesWritten Count	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	All WMQ JMS clients that are or have been connected to this application server	CountStatistic	Low
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	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	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	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	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	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	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	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	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	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	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	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	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
QMAttachedCount	MQLinkStats.QMAttached	Total number of WMQ Queue Managers currently network attached to this application server	6.0	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	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	Alll WMQ Queue Managers that are or have been connected to this application server	CountStatistic	Low

PMI data collection

The PMI data can be collected using the following interfaces:

- Java Management Extension (JMX) interface (J2EE MBeans and WebSphere 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

performance data. PMI data can be gathered from the J2EE managed object MBeans or the WebSphere PMI Perf MBean. While the J2EE MBeans provide performance data about the specific component, the Perf MBean acts as a gateway to the WebSphere 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 Version 6.0. 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 version 4.0.

Enabling PMI data collection

Performance Monitoring Infrastructure (PMI) needs to be enabled (by default PMI is enabled *out-of-the-box*) 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

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 App module at level Medium {M} it enables all the counters at level M, plus all the Low {L} level counters.

WebSphere Application Server Version 6.0 deprecates 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 statistic set.

Though the V5.x monitoring levels {N, L, M, H, X} are deprecated in V6.0, the 5.x PMI APIs are supported for backward compatibility. It is possible to use a V6.0 API to set the monitoring level and a V5.0 API to

get the monitoring level. A new level 'F' – "fine-grained" is introduced to indicate to the V5.x API that the fine-grained or V6.0 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 V6.0 monitoring specification. See "PMI data collection" on page 1629 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.

Enabling PMI using the administrative console

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. When in the **Configuration** tab, settings apply when the server is restarted. When in the **Runtime** Tab, settings apply immediately. Note that you can only enable Performance Monitoring Infrastructure in 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. When in the **Runtime** tab, settings apply immediately. Note that you can only enable the Performance Monitoring Infrastructure in the **Configuration** tab.

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 synchronized update:

Specifies whether access to PMI counters (for example, updates to the counters) is synchronized. There is some performance overhead associated with enabling synchronized 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.

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, V6.0 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, etc., plus the top 35 statistics. This is the default setting.

Extended
Basic setting plus additional WebSphere Application Server components (WLM, Dynamic Cache, etc.). Extended provides key statistics from frequently used WebSphere components.

All Enables all statistics.

Custom
Provides fine-grained control with the ability to enable and disable individual scholastics. Fine-grained configuration is stored in the `pmi-config.xml` file.

Custom monitoring level:

Use this page to enable and disable specific monitoring levels for individual PMI modules.

To view this administrative console page, click **Servers** > **Application Servers** > *server* > **Performance Monitoring Infrastructure (PMI)** > **Custom**.

Custom monitoring level:

Click on the individual PMI module 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.

Enabling Performance Monitoring Infrastructure using the wsadmin tool

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 1631). Make sure to restart the application server.

2. Run the **wsadmin** command, as described in ″Obtaining performance advice from the performance advisors″ in the information center. 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 6.0
 */
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 6.0
 */
public String getInstrumentationLevelString();

/** Return the PMI data in String
    NOTE: This method is deprecated in 6.0
 */
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 6.0
 */
public String getStatsString(ObjectName on, String submoduleName, Boolean recursive);

/** Return the submodule names if any for the MBean
    NOTE: This method is deprecated in 6.0
 */
public String listStatMemberNames(ObjectName on);
```

If an MBean is a StatisticProvider, and if you pass its ObjectName to getStatsObject, you will 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 $perfONameA
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 $perfONameA 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 EJBs, 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 EJBs 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 $perfONname
    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
```

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.

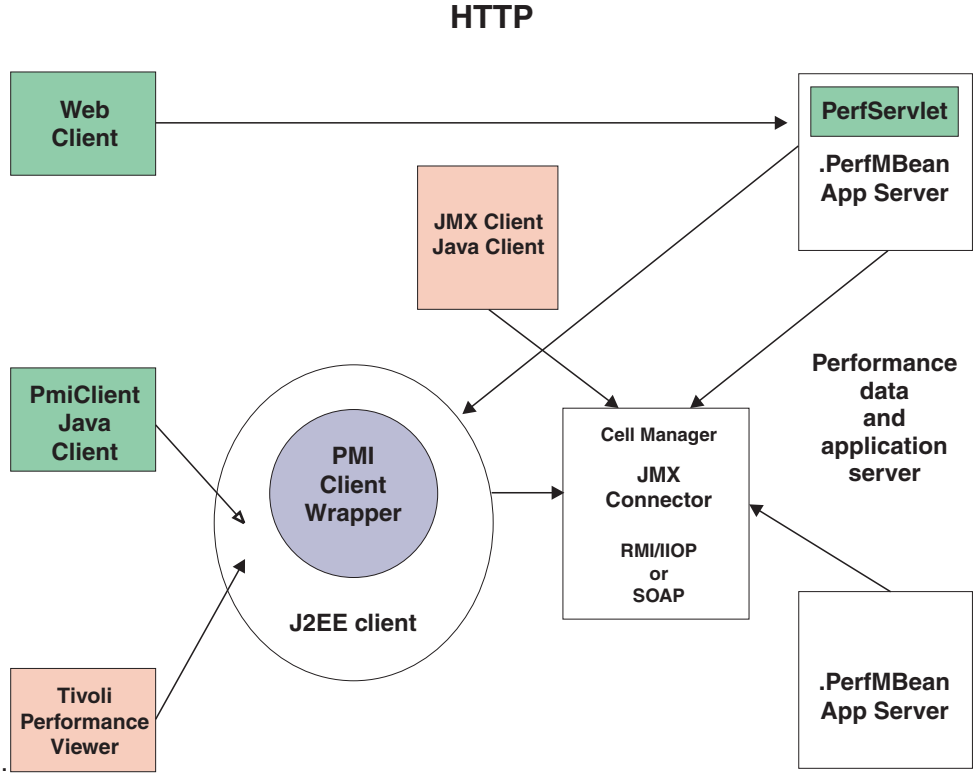
There are three such interfaces - a Java Machine Extension (JMX)-based interface, a PMI client interface, and a servlet interface. All three interfaces return the same underlying data. The JMX interface is accessible through the AdminClient tool as described in “Using the JMX interface to develop your own monitoring application” on page 1655. The PMI client interface is a Java interface. The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.

1. “Using PMI client to develop your monitoring application (deprecated)” on page 1641.
2. “Performance servlet (PerfServlet)” on page 1652
3. “Compiling your monitoring applications” on page 1671
4. “Running your new monitoring applications” on page 1671
5. “Using the JMX interface to develop your own monitoring application” on page 1655.
6. “Developing PMI interfaces (Version 4.0) (deprecated)” on page 1670.

PMI client interface (deprecated)

The data provided by the Performance Monitoring Infrastructure (PMI) client interface is documented here. Access to the 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)

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 ″Obtaining performance advice from the performance advisors″ in the information center..
3. Call the listNodes() and listServers(nodeName) methods to find all the nodes and servers in the WebSphere Application Server domain.
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.

Performance Monitoring Infrastructure client (Version 4.0):

A Performance Monitoring Infrastructure (PMI) client is an application that receives PMI data from servers and processes this data.

In 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 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.

Example: Performance Monitoring Infrastructure client (Version 4.0):

The following is a list of example Performance Monitoring Infrastructure (PMI) client code:

```
/**
 * 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:
 *     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 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:

The following is example code using Performance Monitoring Infrastructure (PMI) client 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) 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.

The PerfServlet 6.0 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 V6.0 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.

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 is deployed in one of the application server instance within the domain. In WebSphere Application Server Network Deployment (ND), the PerfServlet automatically connects to the deployment manager to provide PMI data about the entire cell. 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
```

PerfServlet output: 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 1578 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>
```

Using the JMX interface to develop your own monitoring application

WebSphere Application Server allows you to 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)) {
  perfOName = 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 (perfOName, "setStatisticSet", params, signature);

- getStatisticSet: Returns the current statistic set.
 ac.invoke (perfOName, "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 (perfOName, "setCustomSetString", null, null);

- getCustomSetString: Returns the current custom set specification as a string
 ac.invoke (perfOName, "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(perfOName, "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(perfOName,
"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(perfOName, "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(perfOName,
"getInstrumentationLevel", params, signature);

- getConfigs: get PMI static config info for all the MBeans
            configs = (PmiModuleConfig[])ac.invoke(perfOName, "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(perfOName, "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(perfOName, "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(perfOName, "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(perfOName,
"listStatMembers", params, signature);
```

```
or,

            Object[] params = new Object[]{mbeanSD};
            String[] signature= new String[]
{"com.ibm.websphere.pmi.stat.MBeanStatDescriptor"};
            MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke
(perfOName, "listStatMembers", params, signature);

Refer the Javadoc 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 following is 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 to use AdminClient API directly to get PMI data from PerfMBean
 * and individual MBeans which support getStats method.
 */

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();

    // 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 = "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 = "8880";
        }
        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);
    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();
    }

}

/**
 * Some sample code to set level, get data, and display data.
 */
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
  */
 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
 */
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(perfONname, "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.
 */
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;
    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.getName());
         if(config != null)
             oneStat.setConfig(config);
```

```
        else
            System.out.println("Error: get null config for " + oneStat.getName());
    }
}

/**
 * sample code to show how to get a specific MBeanStatDescriptor
 */
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
 */
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++)
        {
            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
 */
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
 */
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
 */
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");
        return;
    }
    // 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.
 */
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, jvmONname));

                // 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();
        }
    }
}
```

Developing PMI interfaces (Version 4.0) (deprecated)

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

To compile your Performance Monitoring Infrastructure (PMI) code, you must have the following JAR files in your classpath:

- admin.jar
- wsexception.jar
- jmxc.jar
- pmi.jar
- pmiclient.jar
- ras.jar
- wasjmx.jar
- j2ee.jar
- soap.jar
- soap-sec.jar
- nls.jar
- ws-config-common.jar
- namingclient.jar
- management.jar
- pmij2ee.jar

If your monitoring applications use APIs in other packages, also include those packages on the classpath. 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 -extdirs %WAS_HOME%\lib myclass.java
```

Running your new 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 classpath. (The jar files under %WAS_HOME%\lib and %WAS_HOME%\classes folder will be placed in the classpath 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:

```
@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:

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.

The PmiClient communicates with the network manager first, retrieving an AdminClient instance to each application server. When the PmiClient receives the instance, it uses it to communicate with the application

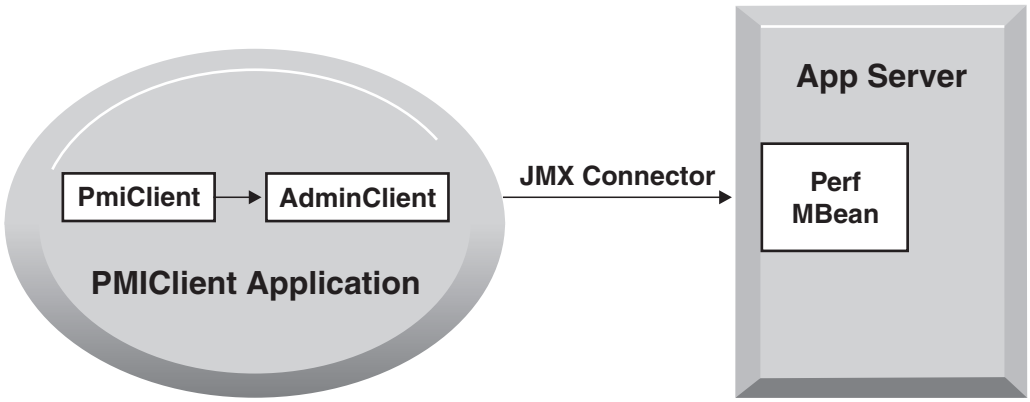
server directly for performance or level setting changes. Since level settings are persistent through PmiClient, you are only required to set it once, unless you want to change it.

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:

In order to run a Performance Monitoring Infrastructure (PMI) client application with security enabled, you must have %CLIENTSOAP% and %CLIENTSAS% properties on your Java virtual machine command line. The %CLIENTSOAP% and %CLIENTSAS% properties are defined in the `setupCmdLine.bat` or `setupCmdline.sh` files.

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 `WAS_ROOT/properties` directory.
2. Set `com.ibm.SOAP.loginUserid` and `com.ibm.SOAP.loginPassword` as the user ID and password for login.
3. 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.

Extending PMI 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. In WebSphere Application Server Version 6.0, PMI has 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.

WebSphere PMI has been extended to allow application developers to 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.

The server side PMI has been extended to allow application developers to 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.

Instrumenting an application with Custom PMI - an example

The following steps are required to instrument an application using Custom PMI:

1. Define 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 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 ();

Monitoring performance with Tivoli Performance Viewer (TPV)

The Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server without leaving the administrative console.

In Version 4.0, Tivoli Performance Viewer was originally named the Resource Analyzer. From TPV, you can view current activity or log Performance Monitoring Infrastructure (PMI) performance data. TPV provides a simple viewer for the performance data gathered by the Performance Monitoring Infrastructure for WebSphere Application Server.

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 and gives recommendations on inefficient settings by using collected PMI data. See ″Obtaining performance advice from the performance advisors″ in the information center.
   - Configure user and logging settings for TPV. These settings can 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 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.
3. View server performance logs. You can 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 store real-time data in log files for later retrieval and analysis.

Viewing current performance activity

You can view the current performance activity of a server using the Tivoli Performance Viewer (TPV) in the administrative console.

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 is affected. To minimize performance impacts, monitor only those servers whose activity you want to monitor.

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**.To start monitoring multiple servers at the same time, select the servers 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 type of data on server activity 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. See ″Using the performance advisor in Tivoli Performance Viewer″ in the information center for additional information.
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 when it detects a long period of inactivity.

Selecting the server to monitor and starting and stopping monitoring:

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**.

Click on any server or node agent to view the current activity for that server.

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 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, 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** (45 seconds) and **Buffer Size** (40 entries) provide you with a 30-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.
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**.

Using the Tivoli Performance Viewer:

Use this page to view and refresh performance data for the selected server, change user 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 on the server name to view the current activity for that server.

View Module(s):

Select one or more servers from the list and press Start monitoring to start the Tivoli Performance Monitor for the selected servers.

Select a resource from the Resource Selection panel, located on the left side, provides a hierarchical (tree) view of resources and the types of performance data available for those resources. Use this panel to select which resources to monitor and to start and stop data retrieval for those resources.

Click the **View Chart** tab in the Data Monitoring panel.

he Data Monitoring panel, located on the right side, enables the selection of multiple counters and displays the resulting performance data for the currently selected resource. It contains two panels: the Viewing Counter panel above and the Counter Selection panel below. If necessary, you can set the scaling factors by typing directly in the scale field.

Refresh:

The refresh operation is a local, not global, operation that applies only to selected resources. The refresh operation is recursive; all subordinate or children resources refresh when a selected resource refreshes. To refresh data:

Click one or more resources in the Resource Selection panel.

Click **File** > **Refresh**. Alternatively, click the **Refresh** icon or right-click the resource and select **Refresh**.

Clicking refresh with server selected under the viewer icon causes TPV to query the server for new PMI and product configuration information. Clicking refresh with server selected under the advisor icon causes TPV to refresh the advice provided, but will not refresh PMI or product configuration information.

Advisor: Select one or more servers from the list and press Stop monitoring to stop the Tivoli Performance Monitor for the selected servers.

User settings: Select one or more servers from the list and press Stop monitoring to stop the Tivoli Performance Monitor for the selected servers.

Log settings: Select one or more servers from the list and press Stop monitoring to stop the Tivoli Performance Monitor for the selected servers.

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, specify `Use synchronized update` in the General properties section of the PMI services panel.

Tivoli Performance Viewer provides the following summary reports for each application server:

Servlets
Servlets show the total number of requests, average response time, and multiplication of total requests by average response time for all the servlets in a table. Servlets provide a sorting feature to help you find which servlet is the slowest or fastest and which servlet is called most frequently.

EJBs Enterprise Java beans (EJBs) show the total number of method calls, average response time, and multiplication of total method calls by average response time for all the enterprise beans in a table. EJBs provide a sorting feature to help you find which EJB is the slowest or fastest and which EJB is called most frequently.

EJB Methods
EJB Methods show the total number of method calls, average response time, and multiplication of total method calls by average response time for the individual EJB methods in a table. EJB Methods provide a sorting feature to help you find which EJB method is the slowest or fastest and which EJB method is called most frequently.

Connection pool
Connection Pool shows a chart of pool size and pool in use for each data source.

Thread pool
Thread Pool shows charts of pool size, active threads, average response time, and throughput in the thread pool.

Performance module: Select the performance module to view performance information.

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.

This article assumes that one or more applications or modules are deployed and running on one or more servers.

You can view TPV summary reports on the following:

Table 25. Types of 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.

Table 25. Types of summary reports (continued)

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.

The default monitoring level enables all reports except the report on EJB methods. To enable the EJB method summary report, configure PMI to use a set of statistics that includes EJB method metrics (All or Custom).

You view TPV summary reports in the administrative console.

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.

Viewing TPV performance modules:

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.

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 it by clicking **–** next to a node.
  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 the check box next to them. By default, the first three counters for each module are shown.
  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**.

- 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 is 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. **Clear Buffer** works by displaying only data with a timestamp newer than the time at which the button was clicked. If **Clear Buffer** is clicked again before the buffer is filled with new data, TPV displays all data currently in the buffer.
- Reset counters to zero (0).
  1. Ensure that one or more modules is selected under **Performance Modules** in the TPV navigation tree
  2. Click **Reset to Zero** beneath the chart or table.

  Some counters report relative values based on how much the value has changed since the counter was enabled. **Reset to Zero** resets those counters so that they report changes in values since the reset operation. **Reset to Zero** also clears the buffer. See ″Clear values from tables and charts″ above for more information about clearing the buffer.

  Counters based on absolute values cannot be reset and are not affected by clicking **Reset to Zero**.

Logging performance data with TPV

The Tivoli Performance Viewer (TPV) provides an easy way to store real-time data for system resources, WebSphere 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 can affect performance and are described in detail in “Using the Tivoli Performance Viewer” on page 1677. The TPV logging feature is not intended to be a full-time monitoring solution.

You can study the sequence of events that led to a peculiar condition in the application server. 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 can corrupt or destroy the performance data stored in the log files.

If monitoring for the node agent is enabled, the log file includes system data from the node agent so that data is available when you replay the log. If monitoring for the node agent is disabled, then that data is not available. By default, TPV starts monitoring the node agent whenever monitoring starts on a server in the node.

You 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 `$WAS_ROOT/profiles/$PROFILE_NAME/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 Network Deployment environment, click the **Browse** button next to the input to browse the various nodes and find the log file to view.
  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.

Tivoli Performance Viewer view logged data:

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 other companies provide performance monitoring, problem determination, and management solutions that can be used with WebSphere Application Server.

These products use WebSphere Application Server interfaces, including Performance Monitoring Infrastructure (PMI), Java Management Extensions (JMX).

See the topic “Performance: Resources for learning” on page 1573 for a link to IBM business partners providing monitoring solutions for WebSphere Application Server.

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.

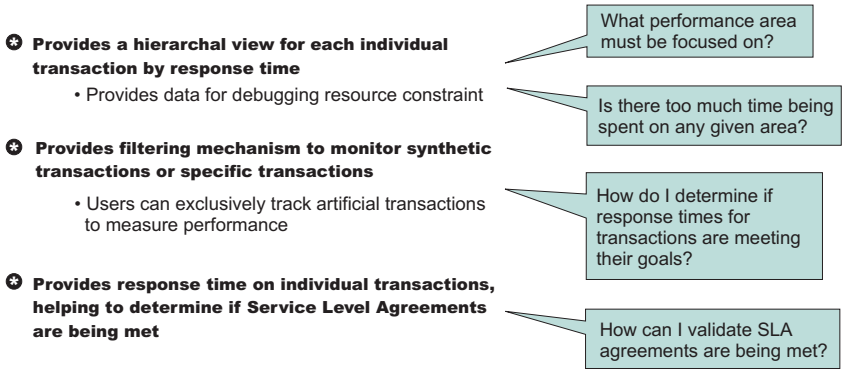
This information includes understanding the application flow that satisfies the end user request. This perspective provides the views of specific servlets that access specific session beans, entity container-managed persistence beans, and a specific database. This perspective is important for the in-depth 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 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 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 backend 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 proactively test 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.

Furthermore, 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 this section, including:

- Detailed explanations about request metrics
- Request metrics process and filters
- Types and formats of output you read

Example: Using request metrics

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 (*plugin_install_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 *install_root*/`profiles`/*profile_name*/`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 *install_root*/`profiles`/*profile_name*/`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:

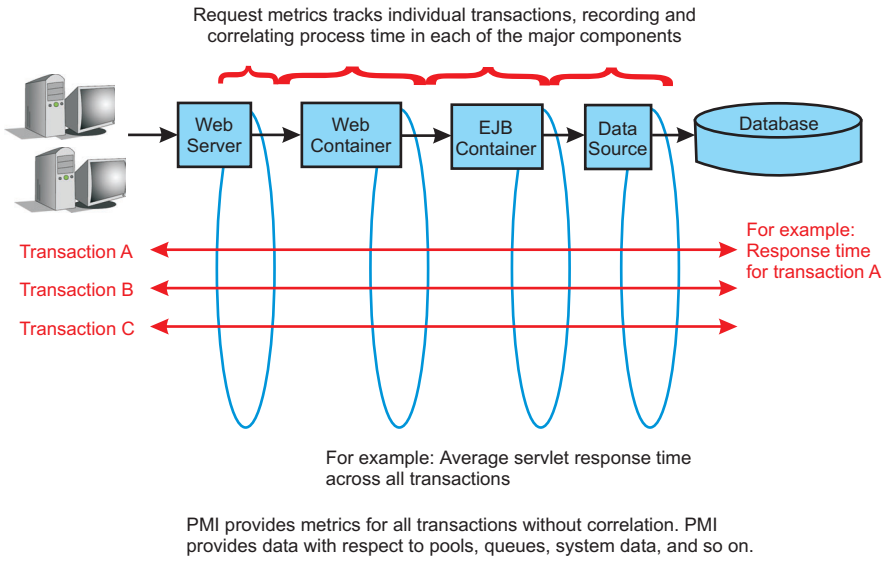
```
PLUGIN:
parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=0
- current:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=1
type=HTTP detail=/hitcount elapsed=90 bytesIn=0 bytesOut=2252

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
```

Understanding the data that 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 1683.

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 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 and Application Server: The Web server plug-in which is only available when using the Web server port, the Web container, the Enterprise JavaBeans (EJB) container, Java DataBase Connectivity (JDBC) calls, Web services (both on the server and the client side), and the Java Message Service (JMS) engine. 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.

Trace records are generated and logged for the Web server plug-in, servlets (Web container), remote EJB calls, JDBC drivers, Web services, JMS requests, and asynchronous beans.

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. Select the **Enable** check box in the **Request metrics** field under the Configuration tab.
4. Specify the components that are instrumented by request metrics.
5. Specify how much data to collect.
6. Enable and disable logging.
7. Enable Application Response Measurement (ARM) Agent.
8. Specify which ARM type to use.
9. Specify the name of the ARM transaction factory implementation class.
10. Isolate performance for specific types of requests.
    a. Add and remove request metrics filters.
11. Click **Apply** or **OK**.
12. 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 1692, 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**.

Request metrics:

Turns on the request metrics feature.

When disabled, the request metrics function is disabled.

Components to be instrumented:

Selects the components that are instrumented by request metrics.

Specify which components; for example, All, servlet, enterprise bean, Java DataBase Connectivity (JDBC), Web services, Java Message Service (JMS), and asynchronous beans are instrumented using request metrics. The default selection is All.

Trace level:

Specifies how much trace data to accumulate for a given transaction.

Including one of the following values:

None No trace.

Hops Generates instrumentation information on process boundaries only (for example, at the entry and exit points for the Web container).

Performance_debug
Generates one additional level of instrumentation data, whereas debug generates detailed instrumentation data.

Debug
Provides detailed instrumentation data, including response times for all intra-process servlet and Enterprise JavaBeans (EJB) calls.

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.

Application Response Measurement (ARM) agent:

Enables request metrics to call an underlying Application Response Measurement (ARM) agent.

To use this feature, ensure that the native libraries of the ARM implementation are present in the *<install_root>*`/bin` directory, and the ARM API Java archive file is present in the *<install_root>*`/lib` directory.

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.

When enabling ARM, include the ARM libraries from the ARM implementation provider in the *<install_root>*`/bin` directory. Type the name of the ARM transaction factory implementation class name that is present in the ARM library that you use in this field.

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 supports ARM 4.0 agent and Tivoli 2.0 ARM agent.

You can choose your own ARM implementation providers to obtain the ARM implementation libraries. Place the ARM API Java archive (JAR) files, found in the `lib` directory of the ARM provider, in the WebSphere Application Server `lib` directory. 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 *<install_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 1573 for more information about the ARM specifications.

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.

If logging time is spent in the Web server, refer to “Regenerating the Web server plug-in configuration file” on page 1692.

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 1692.

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**.

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 the request as it moves through 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 of request metrics filter and enablement for the filter type.

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.

To view this administrative console page, click **Monitoring and Tuning** > **Request metrics** > **Filters** > *filter_type* > **Filter values**.

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) method name request metrics filters.

To view this administrative console page, click **Monitoring and Tuning** > **Request metrics** > **Filters** > *filter* > **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″.

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.

To set the trace level to generate records, perform the following steps:

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**.

The trace level is set to a value you want.

Regenerate the Web server plug-in configuration file as described in the “Regenerating the Web server plug-in configuration file” on page 1692 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

- **Enterprise JavaBeans (EJB) method name filters.** 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 are 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 values for JMS are a combination of a bus name and a destination name.

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 *install_root*/profiles/*profile_name*/logs/*server_name* directory.

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 feature, clear the **Standard Logs** check box.

Enabling or disabling the logging feature is complete.

Viewing performance data from request metrics: 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 1683 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 values select, update, insert or delete for prepared statements. For non-prepared statements, the full statement can display.
- **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`.

On zOS systems when there are multiple servant regions for an application server, there are multiple SystemOut.log files, one for each servant region. Therefore, request metrics might log the trace records in multiple SystemOut.log files. The servant region that handles a request logs the relevant records in its SystemOut.log files. The pid in the current request metrics correlator is the pid for the corresponding servant region. If the system has multiple IP addresses, the IP in the correlator could be one of them, but it should use the same IP for the same servant region.

Extending request metrics

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

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");

//update
try {
 out.println(atxwrp.update());
} catch (Exception e) {
 out.println(e);
}
//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;
}

}
```

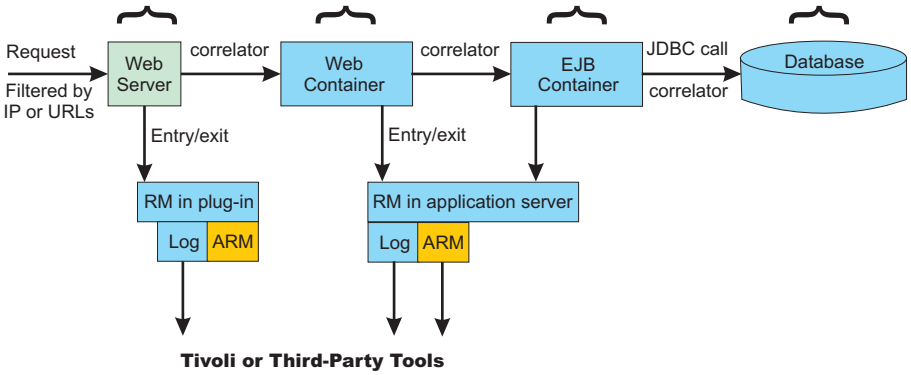
There are several potential scenarios for using the PmiRmArmStack. This example shows a scenario where code accesses an existing PmiRmArmTx on the stack, extracts the correlator, and calls blocked and unblocked. This is a typical scenario when sending a correlator along an unsupported protocol. In this 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  }
```

Understanding the 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 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 components to provide an end-to-end picture of the transaction, as shown in the following diagram:



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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and service marks

For trademark attribution, visit the IBM Web site http://www.ibm.com/legal/us/.